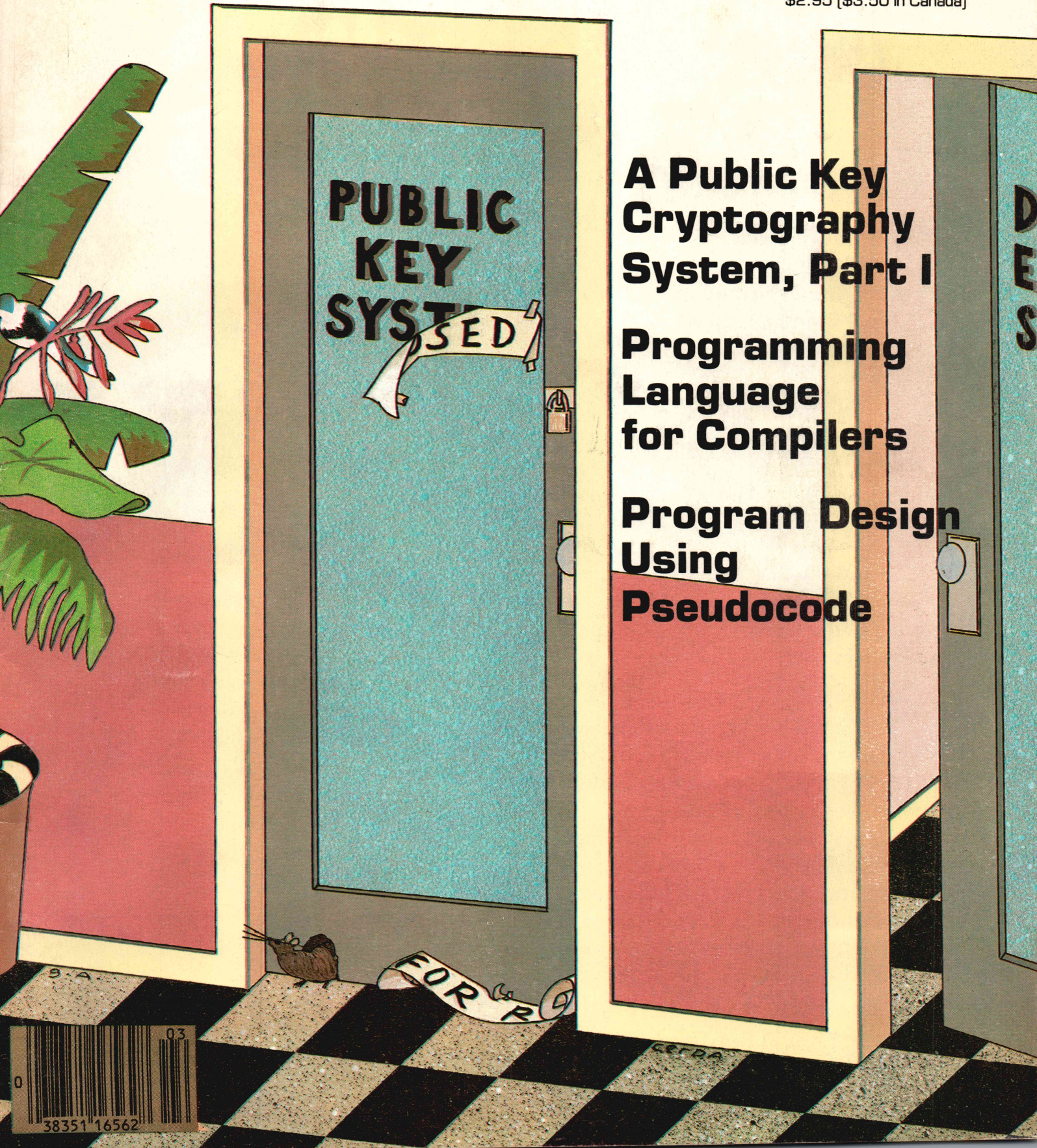


Dr. Dobb's Journal

For the Experienced in Microcomputing

#89 March 1984

\$2.95 [\$3.50 in Canada]



**PUBLIC
KEY
SYSTEM**

**A Public Key
Cryptography
System, Part I**

**Programming
Language
for Compilers**

**Program Design
Using
Pseudocode**

FOR



"dBASE II® is far, far better than a shoehorn."

*Rusty Fraser
President
Data Base Research Corp.*

"We laughed when our customers asked us to put our minicomputer-based real-time accounting system, The Champion,™ on a micro.

"No way was it going to fit, we thought.

"We'd have to create our own database management system and, even then, it'd be a tight squeeze.

"Then we discovered dBASE II, the relational database management system for microcomputers from Ashton-Tate."

"dBASE II was a perfect fit."

"dBASE II is a program developer's dream come true. The dBASE II RunTime™ module quickly provided us with the powerful text editing, data entry speed and other 'building block' capabilities we needed to develop and deliver a new Champion to our customers—the leading real-time on-line accounting system available for a micro."

The short cut to success.

The dBASE II RunTime module has helped a lot of program devel-



opers like Data Base Research become successful software publishers.

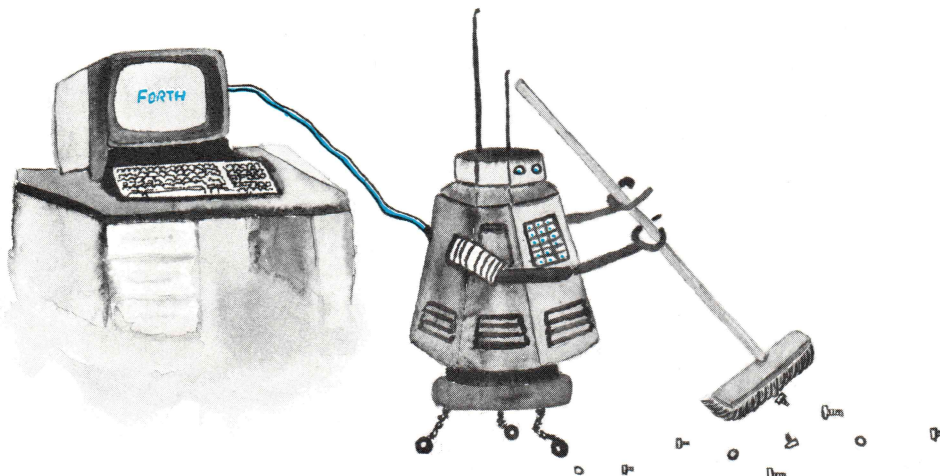
For more about dBASE II and RunTime, contact Ashton-Tate 10150 West Jefferson Boulevard, Culver City, CA 90230, (800) 437-4329, ext. 217. In the U.K., call (0908) 568866.

For more about The Champion, call Data Base Research at (303) 987-2588.

ASHTON · TATE

dBASE II and RunTime are registered trademarks of Ashton-Tate.
The Champion is a registered trademark of Data Base Research Corporation.
©Ashton-Tate 1983.

FORTH GIVES YOU TOTAL CONTROL



GRAPHICS • GAMES • COMMUNICATIONS ROBOTICS • DATA ACQUISITION • PROCESS CONTROL

FORTH: for Z-80®, 8080, 8086, 68000, and IBM® PC
(Complies with the New 83-Standard)

FORTH programs are instantly portable across the four most popular microprocessors.

FORTH is interactive and 20 times faster than BASIC.

FORTH programs are highly structured and easy to maintain.

FORTH provides direct control over all interrupts, memory locations, and i/o ports.

FORTH allows full access to DOS files and functions.

FORTH application programs can be distributed as turnkey COM files with no license fee.

FORTH Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Custom programming, consulting, and educational services available.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, de-compiler, utilities, and detailed technical manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, \$50.00; **8080 FORTH** for CP/M 2.2 or MP/M II, \$50.00; **8086 FORTH** for CP/M-86 or MS-DOS, \$100.00; **PC/FORTH™** for PC-DOS, CP/M-86, or CCPM, \$100.00; **68000 FORTH** for CP/M-68K, \$250.00

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly for programs and data.

PC FORTH + \$250.00
8086 FORTH + for CP/M-86 \$250.00
68000 FORTH + for CP/M-68K \$400.00

FORTH Cross Compiler allows you to customize the FORTH nucleus, recompile on a host computer for a different target computer, generate headerless and ROM-able code. Supports forward referencing. Produces executable image in RAM or disk file. No license fee for applications, \$300.00.

FORTH Native Code Compilers

For Z80 FORTH and CP/M \$100.00
For 8086 FORTH and CP/M-86 \$200.00
For IBM PC and PC-DOS \$200.00

Extension Packages

Software floating point (Z-80, 8086, PC only), \$100.00; AMD 9511 support (Z-80, 8086, 68000 only), \$100.00; Intel 8087 support (8086, PC only), \$100.00; Advanced color graphics (PC only), \$100.00; Symbolic interactive debugger (PC only), \$100.00; PC/TERM Communications/file transfer for Smartmodem, \$60.00; Cross reference utility, \$25.00; PC/GEN (custom character sets, PC only), \$50.00; Curry FORTH Programming Aids, \$150.00; B-Tree index manager, \$125.00; B-Tree index and file manager, \$200.00; QTF + Screen editor for IBM PC, \$100.00; Quad Integer Math Pack, \$25.00.

AUGUSTA, Ada subset compiler from Computer Linguistics for Z-80 CP/M 2.2 systems, \$90.00

"Starting FORTH" tutorial by Brodie, soft-cover, \$16.00.

INTEL 8087-3 Numeric Coprocessor, \$250.00

Z-80 and 8080 **FORTH** require 48 Kbytes RAM. 8086 and 68000 **FORTH** require 64 Kbytes RAM. Disk formats available include: 8" standard CP/M SSSD, Northstar 5 1/4" QD, Kaypro 5 1/4", Apple 5 1/4", Micro-Mate 5 1/4", MS-DOS 5 1/4", Osborne 5 1/4", DD, and Sage. Most other formats can be special ordered. Dealer inquiries invited.

Z-80 is a registered trademark of Zilog, Inc.; CP/M is a registered trademark of Digital Research, Inc.; IBM is a registered trademark of International Business Machines Corp.; Augusta is a trademark of Computer Linguistics; PC/FORTH and PC/GEN are trademarks of Laboratory Microsystems Inc.



Laboratory Microsystems Incorporated
4147 Beethoven Street, Los Angeles, CA 90066
Phone credit card orders to (213) 306-7412



Circle no. 27 on reader service card.

WHY PROFESSIONALS CHOOSE AZTEC C

SOFTWARE SYSTEMS

- AZTEC C is the most complete implementation of UNIX V7 'C' available for microcomputers.
- AZTEC C is portable. 'C' code can be freely moved between UNIX V7 and microcomputer systems. AZTEC C is available for PC DOS (MSDOS), CP/M 86 (MP/M 86), CP/M 80 (MP/M 80), APPLE DOS, MODEL III (IV), COMMODORE 64, and ATARI. All versions of AZTEC C are source compatible.
- AZTEC C is a complete development system that includes relocating assembler, linkage editor, library utility, debugging aids, overlay support, interfaces to Microsoft and Digital Research development software, and run time routines for I/O, utility, and scientific functions. Source for all library routines is provided.
- AZTEC C generates fast native code for the 6502, 8080, Z80, 8088, and 8086.
- Cross compilers are available from PDP-11, 8086, 8080, and 6502 processors.
- Since its release in 1981, AZTEC C has been acquired by several thousand users. Commercial applications include a wide variety of business, scientific, word processing, database, entertainment and financial applications.

PRICE LIST

AZTEC C86	PC DOS (MSDOS)	\$249
AZTEC C86	CP/M 86 (MP/M 86)	249
AZTEC C86	CP/M 80 (MP/M 80)	199
AZTEC C-11	APPLE DOS	199
AZTEC C65	COM 64 OR ATARI	199
AZTEC Cross Compilers	PDP-11	2,000
AZTEC Cross Compilers	Other	500

Order by phone or mail. Specify product and disk format. Check, Money Orders, COD, VISA, MC, and purchase orders are acceptable. NJ add 6% sales tax.

MANX SOFTWARE SYSTEMS
Box 55, Shrewsbury, NJ 07701
(201) 780-4004
(201) 530-7997
(201) 530-7708

Order phone:
Tech information:
Tech support:

Shipping: COD, 2nd day delivery, or Canada, add \$5. Canada 2nd day or US next day delivery, add \$20. Outside North America, add \$20, and for 2nd day add \$75.

Circle no. 31 on reader service card.

FORTH PERFORMANCE

The Fourth Generation Comes of Age

MODAL DTC

Finally you can utilize the power of Forth with efficient performance.

Modal DTC is an interactive, extensible **Direct Threaded Code** software environment. Forth applications execute **4-5 times faster** when loaded under Modal DTC. For example, Modal DTC runs the **Forth Sieve Benchmark** in 1.7 secs/pass on a 4MH Z-80 with no wait states! You get the performance of the fastest compilers in a sophisticated fourth generation environment.

Package Includes:

- ▲ File Management Functions
- ▲ Full-Featured Screen Editor
- ▲ Symbolic Debug Facility
- ▲ Memory-Mapped Video
- ▲ Forth Vocabulary
- ▲ Conditionals in Open Code
- ▲ Conditional Compilation
- ▲ Documented Demo Package
- ▲ Detailed Operation Manual
- ▲ Professional Support and Upgrade Policy.
- ▲ Assembler with Conditional Assembly and Macro Capabilities
- ▲ Many Extensions Under Development.

Modal DTC/80 Release 1.1 for CP/M 8080/8085/Z-80 64K Memory. Available in IBM 8" and most 5.25" disk formats. Very fast video on Osborne and similar systems. Specify computer and disk format when ordering. Personal License \$149.⁰⁰ US.

Available soon for CP/M-86 and PC DOS Systems

modal systems

3030 Tanglely Ave.
Houston, TX 77005
(713) 660-7394

VISA/MC/MO/COD

Circle no. 36 on reader service card.

Dr. Dobb's Journal

For the Experienced in Microcomputing

March 1984
Volume 9, Issue 3

CONTENTS

ARTICLES

16 RSA: A Public Key Cryptography System, Part I

by C. E. Burton

This two-part article details implementation of the Rivest-Shamir-Adleman Public Key cryptography system on a microcomputer. This first installment includes a quick overview of the author's implementation language and a discussion of the mathematical algorithms used. Next month, Part II will describe the generation of the keys and the encryption/decryption system, and the use of digital signatures. (Reader Ballot No. 192)

44 Introduction to PL/C: Programming Language for Compilers

by Morris Dovey

Because development of new and more powerful processors seems to be outstripping that of supporting software, the PL/C programming language was designed to fill the need for rapid, low-cost development of reliable programming language compilers. Included here are discussion of the language structure and syntax, and a listing of the compiler in PL/C and macro-assembly code. (Reader Ballot No. 193)

70 Program Design Using Pseudocode

by Ken Takara

Program design can be tedious, even using some of the available design tools. This article uses the construction of a "shoot-em-up game" to examine the flexibility and convenience that using pseudo-code can provide in the design process. (Reader Ballot No. 194)

78 More On Binary Magic Numbers

by Dale Wilson

Inspired by Edwin Freed's "Binary Magic Numbers" (DDJ No. 78, April 1983), this article provides a C language implementation of a number of the algorithms presented by Freed. This provides both useful functions and interesting examples of the use of C to implement them. (Reader Ballot No. 195)

DEPARTMENTS

6 Editorial

7 Letters

9 Dr. Dobb's Clinic

Basically Precise; Transcendently Precise; Stocking Up (Reader Ballot No. 190)

12 CP/M Exchange

How fast is CP/M Plus?; CP/M 2.2 BIOS Function: SELDSK; CP/M V2.2 Application Note (Reader Ballot No. 191)

84 Software Reviews

SXR PLUS: Sorted Cross Reference; Perfect Writer/Perfect Speller

88 Book Reviews

Cryptography: Proceedings of Workshop on Cryptography; IBM Data Files; Microprocessor Support Chips: Theory, Design, and Applications; Z80 Applications; Concepts for Distributed Systems Design; An Introduction to Numerical Methods with Pascal; Augmented Transition Networks

92 16-Bit Software Toolbox

The collected, sorted and tabulated responses to last September's floating-point benchmark (Reader Ballot No. 196)

98 Of Interest

(Reader Ballot No. 197)

102 Advertiser Index

Publisher – Jane Nissen Laidley

Editorial

Editor – Reynold Wiggins

Managing Editor – Randy Sutherland

Contributing Editors –

Robert Blum, Dave Cortesi,

Ray Duncan, Anthony Skjellum,

Michael Wiesenber

Marketing

Marketing Director – Beatrice Blatteis

Marketing Assistant – Sally Brenton

Marketing Coordinator – Ed Gueble

Advertising

Advertising Director – Beatrice Blatteis

Advertising Sales –

Alice Hinton, Walter Andrzejewski

Circulation

Circulation Director – Terri Pond

Circulation Assistant – Frank Trujillo

Production

Art Director/Production Manager –

Barbara Ruzgerian

Production Coordinator –

Shelley Rae Doeden

Production Assistant – Alida Hinton

Typesetter – Jean Aring

Cover Illustration – Gregory A. Cerda

Copyright © 1984 by People's Computer Company unless otherwise noted on specific articles. All rights reserved.

Subscription Rates: \$25 per year within the United States; \$44 for first class to Canada and Mexico; \$62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Donating Subscribers Contributing Subscriber: \$50/year (\$25 tax deductible). **Retaining Subscriber:** \$75/year (\$50 tax deductible). **Sustaining Subscriber:** \$100/year (\$75 tax deductible). **Lifetime Subscriber:** \$1000 (\$800 tax deductible). **Corporate Subscriber:** \$500/year (\$400 tax deductible, receives five one-year subscription).

Contributing Subscribers: Christine Bell, W. D. Rausch, DeWitt S. Brown, Burks A. Smith, Robert C. Luckey, Transdata Corp., Mark Ketter, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor Jr., Ted A. Reuss III, Info World, Stan Veit, Western Material Control, S. P. Kennedy, John Hatch, Richard Jorgensen, John Boak, Bill Spees, R. B. Sutton. **Lifetime Subscriber:** Michael S. Zick.

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines and the current compensation schedule by sending a self-addressed, stamped envelope.

Foreign Distributors UK & Europe: Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; Euro Computer Shop, 182 Rue du Faubourg St. Denis, 75010, Paris, France; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – premiere, F-75014, Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** AXCH Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. (Write for Canadian Distributors)

EDITORIAL

Several people found the time interval between the formal announcement of the telecommunications issue in our pages and the copy deadline to be a bit tight. Let that not be said about our *next* special issue. Be advised that our annual Forth issue is again planned for September, and the copy deadline will be May 14, 1984. We will provide more details next month, but interested parties may contact us at P. O. Box E, Menlo Park, CA 94026; (415) 323-3111.

We have a correction and an elaboration regarding last month's issue. On page 44, a telephone number was provided for testing one's VPC implementation. Unfortunately, the number given was the main number for Unir, not the VPC test line. The correct number to dial to test your VPC is (317) 842-6986.

The sidebar which accompanied the communications protocols article noted that a Monte Carlo simulation was being set up to estimate some of the accuracies for various error-checking schemes. Author Leslie Brooks has provided a Letter to the Editor this month which details the outcome of the simulation, and its effect on their initial predictions.

Below you will find the list of our current referees. As with other aspects of *DDJ*, we assume that the board will evolve over time. While we intend to list the referees that work on each issue, we will also publish a complete list periodically.

Our appreciation to those who have all along been informally providing technical advice and insights. Our special thanks to David E. Cortesi for his continuous and substantial support, and to Kim Harris for his willingness to look at so many Forth articles over the years.

Reynold Wiggins

DDJ's Current Board of Referees

Dennis Allison, Computer Science Dept.,
Stanford University

Ian Ashdown, Professional Engineer

Joe Barnhart

S. M. Bellovin, AT&T Bell Laboratories

Robert Blum, Contributing Editor, *DDJ*

Patrick Burnstad, Lands End Computers,
PATCA, SVCS, IEEE

Wayne Chin, Hewlett-Packard

David D. Clark, Department of Chemistry,
Pennsylvania State University

David E. Cortesi, Contributing Editor, *DDJ*

Keith Coye, President Consulting Services
Corporation and PicoNet

Mel Cruts, Diagnostic Engineer, Trilogy

Ray Duncan, Contributing Editor, *DDJ*

Michael T. Enright, Cal Omega Inc., ACM/
SigGraph, IEEE

Z. Z. Fisher, Data Communications Consultants

Jim Fleming, Unir Corporation

Dr. Georges Grinstein, Computer Science
Dept., Fitchburg State University

A. Gomez, Telecom, Inc.

Kim Harris, Forthright Enterprises

J. E. Hendrix

William P. Hogan, Owner, William Hogan Associates,

David W. Hopper, Senes Consultants Limited

J. C. Johnston, Graduate Student Cleveland
State University

Michael P. Kelly, Design Software

John P. Keyes, Student Syracuse University

David Kirkland, Baker & Botts

Richard G. Larson, Department of Mathematics,
Statistics, and Computer Science, University
of Illinois at Chicago

Ben A. Laws, Assistant Professor, Computer
Science, North Texas State University

Mohamed el Lozy, M.D., Ph.D., Harvard
School of Public Health

Patrick Lynch, Tandem Computers Inc. of
Cupertino

Clay Phipps, ACM

William Ragsdale, President, Forth Interest
Group

Donald L. Rastede

Richard A. Relph

John A. Rogers, Fortune Systems

Douglas W. Rosenberg, Optimal Software

David Ross, Professor, University of Iowa

Darryl E. Rubin, ROLM Corporation

Joseph Sharp, Micro Science Associates

Henri Jonathan Socha, SochaLogical Research,
IEEE, ACM

John K. Taber, IBM

Scott D. Thomas, Informatics General Corp.

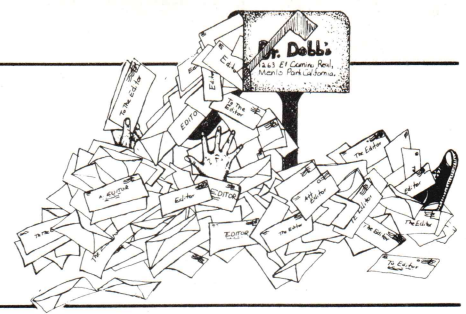
Allen Tigert, Kontron Medical Electronics

Bob Wheeler, Custom Hardware

Charles Wilde, Aton International Inc., IEEE,
ACM

Dr. Dobb's Journal (USPS 307690) is published twelve times per year by People's Computer Company, P.O. Box E, Menlo Park, CA 94026. Second class postage paid at Menlo Park, California 94026 and additional entry points. Address correction requested. Postmaster: send form 3579 to People's Computer Company, Box E, Menlo Park, CA 94026 • 415/323-3111

ISSN 0278-6508



The editorial response card is a great way to talk to us, but don't forget that Dr. Dobb's Journal also welcomes letters to the editor as a forum for ideas, innovations, irascibility and even idiosyncrasies. Some letters may be edited for clarity and brevity. The Doctor likes hearing from you — keep on writing.

Method:	XOR	Sum	Sum w/carry
Random Errors:	1 in 136	1 in 187	1 in 221
Clustered Errors:	1 in 417	1 in 381	1 in 360

Table
Probability of Missing an Error

Accuracy Update

Dear Doctor:

In our recent article on communications protocols (February 1984, *DDJ* No. 88), my friend John Rasp and I were discussing the relative merits of various methods of error detection. As we said there, the mathematics becomes *very* tricky if the changed bits are clustered together rather than randomly distributed. In order to get accurate results for the clustered errors, and to check our results for random errors, we finally ran a Monte Carlo simulation on the university's Cyber 760 mainframe. The results were very enlightening; although he couldn't prove it John had not expected the two cases to differ, but in fact they did. It turns out (as you can see by looking at the table top right) that all of the common methods of error detection are significantly better at catching errors if the changed bits are clustered together. In fact, the XOR method of calculating a checksum is the *worst* method of calculating the checksum if the errors are random, but the *best* method if the errors are clustered.

These results should be accurate to 3 digits. The simulation ran 40,000 iterations, generating a block of 128 bytes of random data each time, then randomly choosing the first bit to be clobbered, then randomly choosing the number of bits to be clobbered, then choosing the values of the clobbered bits. The number of bits to be hit was clustered around 10, in the usual bell curve. The simulation was written in Fortran, and required 15 minutes of CPU time to run (on a 10 megaflop machine)!

I hope this clears up any lingering questions.

Sincerely,
Leslie Brooks
Computing Center
The Florida State University
Tallahassee, FL 32306

(Continued on page 90)

SEATTLE GIVES YOU AN EDGE IN S-100 SYSTEM DESIGNS

You can unlock new system capabilities with high-performance S-100 boards from Seattle Computer. All are IEEE-696 compatible. But, for innovative systems that demand performance beyond the limits of conventional S-100 boards, you'll want to know more about these Seattle Computer products. For example, with our 8 MHz 8086 CPU, you'll be able to build systems that run faster and consume less power than before. Take a closer look:

8086 CPU Set: 8 MHz 8086 CPU • CPU Support board includes a console serial port, a second serial port, Centronics parallel port, vectored interrupt controller, four 16-bit timers and EPROM monitor for 8086 • MS-DOS 2.0 plus development utilities • 8087 numeric coprocessor is optional
• **Single Qty: \$595.00**

64k Static RAM Fully static design makes interfacing easy • Compatible with a variety of CPU and DMA devices • High-speed (85 ns) RAMs operate to 10 MHz with no wait states • 16k, 32k, and 48k OEM versions are available
• **Single Qty: \$495.00 (64k)**

Disk Master™ Controls as many as four 8" and four 5.25" floppy disk drives simultaneously, in any combination • Uses 1793 disk controller chip • Can be used with 10 MHz CPUs • **Single Qty: \$325.00**

Multi-Port Serial Card 2- and 4-port versions are available • These RS-232 ports operate as either "data sets" or "data terminals" • 36" cables included
• **Single Qty: \$280.00 (4-port)**
\$210.00 (2-port)

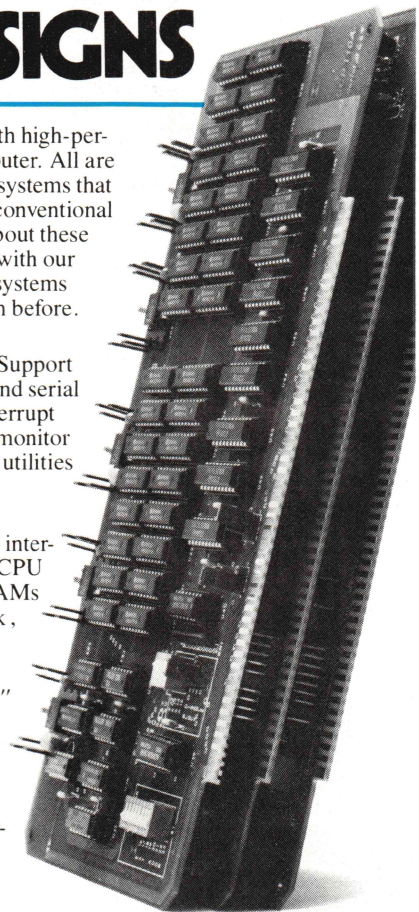
For the whole story on high-performance Seattle Computer S-100 boards, call:

1-800-426-8936

Dealer and OEM inquiries are invited.

**SEATTLE
COMPUTER**

1114 Industry Drive
Seattle, WA 98188



SOLUTIONS INFINITE

Like the molecules in a snowflake, the elements of a computer database can be structured and related in an infinite variety of combinations. Being able to present these combinations quickly and efficiently, with maximum flexibility and minimum programming knowledge, is the mark of excellence which sets the sophisticated database management system apart from the ordinary.

Thousands of satisfied users, from amateur and professional programmers to government agencies, major corporations and industries, have found that DataFlex has no equal in applications development software.

Write or phone for our latest literature and a list of existing applications.

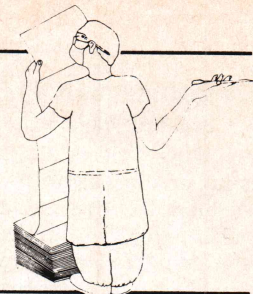
Compatible with CP/M, CP/M-86, MP/M-86, MSDOS, PCDOS, TurboDOS, Novell Sharenet, PC-Net, Molecular N-Star, Televideo MmmOST, Action DPC/OS, Omninet, IBM PC w/Corvus and OSM Muse.

DATA FLEX
APPLICATIONS DEVELOPMENT SOFTWARE

DATA ACCESS CORPORATION

8525 S.W. 129th Terrace, Miami, Florida 33156 (305) 238-0012 TELEX 469021 Data Access CI
MSDOS is a trademark of Microsoft. CP/M and MP/M are trademarks of Digital Research. DataFlex is a trademark of Data Access Corp.

Circle no. 17 on reader service card.



by D.E. Cortesi, Resident Intern

Basically Precise

That describes the material sent us by Allan Behler, printed in the November Clinic. Allan had discovered that, in Microsoft BASIC-80, a constant assigned to a double-precision variable wasn't stored as expected:

```
DEFDBL A
A=134.12
PRINT A
134.1199951171875
```

The display could be made correct with PRINT USING, but repeated use of such values could accumulate significant errors. With a good deal of difficulty he worked his way to a solution using a rounding function to convert from a single-precision value to a double-precision one. Here is a corrected, improved version of the rounding function shown here in November:

```
10 DEF FNRD#(X#)=
    INT(X#*100+0.5)/100#
20 DEFDBL A
30 A=134.12 : PRINT A
40 A=FNRD#(A) : PRINT A
RUN
134.1199951171875
134.12
```

Three things are needed to make the function work as expected. First, there are two expressions in FNRD# (one is the argument of INT, the other is the division), and at least one of the elements of each expression must be double precision. Why? Read the following rule from the *BASIC-80 Reference Manual* (Microsoft, 1979, page 1-8):

"During expression evaluation, all of the operands in an arithmetic or relational operation are converted to the same degree of precision, i.e., that of the most precise operand."

In other words, if the divisor isn't "100 #" indicating a double-precision constant, the result of the division will be single precision. The single-precision result will be stretched to double to satisfy the "#" in the name of the function, and the result will be exactly the same as assigning 134.12 to A in the first place.

Second, the function name has to indicate that it returns a double-precision result. If it doesn't — if its name is simply FNRD, for example — the double-precision result of the expression will be truncated to single precision to match the implicit type of the function. Then *that* result will be stretched to double precision

for the assignment to A, and we are back to square one.

All these problems are the effects of another rule (*Ibid*, page 1-9):

"If a double-precision variable is assigned a single-precision value, only the first seven digits, rounded, of the converted number will be valid. [Note: it's six digits in the IBM manual.] The absolute value of the difference between the printed [sic] double-precision number and the original single-precision value will be less than 6.3E-8 times the original"

It isn't clear to us why, after a constant has been converted correctly to a single-precision float value, copying it to a double-precision float value should change its low-order bits (if that's what happens). However, the effect is there, it's documented, and it's dangerous.

Finally, the function can still fail if it doesn't indicate that it requires a double-precision argument. If its dummy argument is named X rather than X#, and if the value of the argument has more than six significant digits, the result is wrong. The (effectively double-precision) argument gets truncated to single precision when assigned to the dummy variable, and data is lost.

Well, Allan's record of his explorations provoked some sharp replies. Let's read some of them. Charles Marshall says, "If the example is rewritten to use a double-precision constant, the correct answer is printed:

```
A=134.12# : PRINT A
134.12
```

"Mr. Behler continues to use mixed-mode arithmetic in the three program segments, which will tend to exaggerate the problem he is trying to solve. In addition his use of 0.51 as a rounding constant is invalid, resulting in, for example, the number 1.0049 being 'rounded' to 1.01."

Rounding is another whole topic. We think Marshall is right, but would anyone care to guess why Allan might have (very deliberately) used 0.51 instead?

Meanwhile, Joseph McDermott writes:

"People have been stumbling over this problem since 1977. I found the correct solution on my TRS-80 in 1978, and it still applies to the IBM PC. Why does not Microsoft explain

it is their manuals and save everybody so much grief? Numeric precision problems disappear if a few simple rules are followed."

McDermott's Rule #1 is: *All values entering a calculation must be double precision.* You must be consistent from beginning to end of your program, *no exceptions*, he says. This extends to things like input where, if you read numeric input as a string and then convert it, you must remember to append the magic pound-sign:

```
INPUT NUM$
NUM# = VAL( NUM$ + "#" )
```

"BASIC reverts to single precision at the slightest excuse," he says. "A double-precision variable on the left of the assignment statement is *not* sufficient to enforce double-precision calculations."

Joseph Sabin wrote in with what we think is a false lead: "Whenever you change a single-precision number to a double-precision number, you must use the double-precision exchange function:

```
A# = CDBL(123.12)
```

or you will get inaccurate results."

A good point, we thought, until we tried it:

```
A# = CDBL(134.12)
PRINT A#
134.1199951171875
```

Oops. Sorry, Joseph, we can't find any difference between using CDBL and not. It is definitely not a replacement for the use of a double-precision marker on the constant itself — "134.12#" and "CDBL(134.12)" do not appear to be the same thing. In fact, the documentation for CDBL reveals exactly that. Take this example, exactly as shown in the IBM BASIC manual (first edition, page 4-31):

```
10 A = 454.67
20 PRINT A; CDBL(A)
RUN
454.67 454.6699829101563
```

That's Behler's original problem; CDBL does *not* do what FNRD# does. What FNRD# does is probably not useful when working with constants; the trailing pound sign does it better. Its only use would be in the rare cases when you have a computed value of money in a single-precision variable and want to assign it to a double-precision variable. In that specific case it does a better job than CDBL.

Transcendently Precise

Richard Falk writes asking for help in finding, "either a machine code source listing or a fast algorithm for trig functions on the 6502. I've coded the power series for sin, cos, tan, arcsin, and arctan, but these require so many terms for sufficient accuracy (12-13 decimal digits) that they run too slowly." He notes that the best sin function he's come up with takes 500 ms to execute, compared to 340 ms for the BASIC sin function on the same machine. Would anyone care to recommend a good reference for this? We suppose that Falk would like a cookbook approach, rather than a course in numerical analysis.

Stocking Up

Dick Mesirov is the kind of reader we really appreciate: He sent us some original, unsolicited input — a "sponse," not a re-sponse. (Go thou and do likewise!) Here's how he tells it:

"I am a market maker on the Philadelphia Stock Exchange Options

floor. If you've seen TV shots of those guys screaming, hollering, and waving their arms in the commodity pits, well, that's what I do. I store data on the stocks I trade: opening price, high, low, close, and the direction of last trade (up or down). This data is entered and used daily, and I store it by month. That is, all of October's entries are stored in the file named xxxOCT83, where xxx is the three-letter symbol for the stock.

"When I run the program each evening, after entering the day's data, I recall several months' data in addition to the current month's. The number of months required varies depending on what I am trying to do. Originally I recalled each month by name; then I came up with the routine listed here. It recalls and saves one month's data a day at a time, then automatically steps down by one month and recalls the prior month's data. After getting January's data it also steps down one year.

"I've never seen anything like it in any of the books I have nor in any of the commercial programs I've looked at. The same approach could be used by day or I guess for any series that can be listed as a string."

Dick's routine appears in Listing One (below). The original was cleanly structured using IFs and GOTOs; we took the liberty of recoding it to use nested FOR and WHILE loops to emphasize its shape. The only problem we can see is not in the program but in CP/M. There is a definite limit to the number of files you can store on a disk. Typically there are only 64 or 128 entries in the disk directory. When Mesirov accumulates a year's data on ten stocks he can look forward to getting a file error from filling up the directory. Within that limit, this looks like a useful technique. ■■■

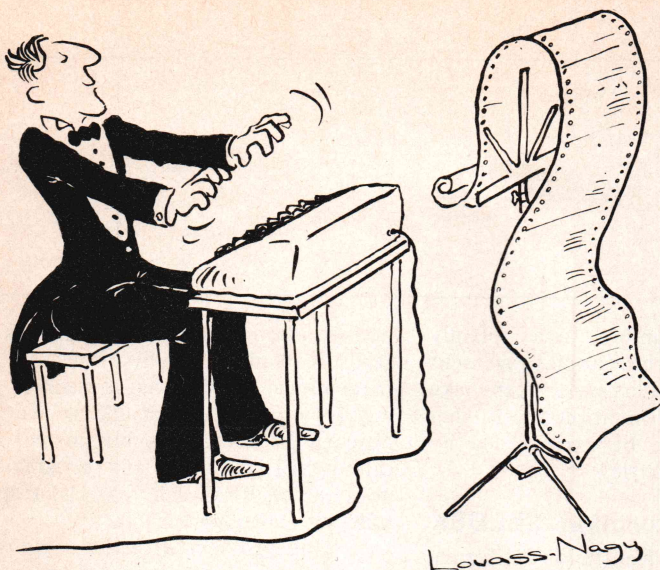
Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 190.

Dr. Dobb's Clinic (Text begins on page 9)

```
9000 REM
9010 REM   SUBROUTINE TO READ STOCK HISTORY FILES
9020 REM history files are named SSSMMYY, where
9030 REM   SSS=stock-id, MMM=month, YY=year
9040 REM returns N=number of entries read
9050 REM
9060 IM$="JANFEBMARAPRMAYJUNJULAUGSEP OCTNOVDEC"
9070 INPUT "NAME OF STOCK";ST$
9080 INPUT "CURRENT MONTH";MO$
9090 B=INSTR(IM$,MO$)
9100 INPUT "CURRENT YEAR";YC$ : YC=VAL(YC$)
9110 INPUT "HOW MANY MONTHS' DATA NEEDED";NM
9120 N=0
9130 FOR JM=1 TO NM
9140   REM read one file's (month's) data
9150   OPEN "I",#1,ST$+MO$+YC$
9160   WHILE NOT EOF(1) : N = N+1
9170     REM here READ a stock entry into various
9180     REM ..arrays subscripted by N
9190   WEND : CLOSE
9200   REM get the prior month and year
9210   B=B-3
9220   IF B<1 THEN B=B+36 : YC=YC-1 : YC$=RIGHT$(STR$(YC),2)
9230   MO$=MID$(IM$,B,3)
9240 NEXT JM
9250 RETURN
```

End Listing



•NEW PRODUCTS•

Before Johann Sebastian Bach developed a new method of tuning, you had to change instruments practically every time you wanted to change keys. Very difficult.

Before Avocet introduced its family of cross-assemblers, developing micro-processor software was much the same. You needed a separate development system for practically every type of processor. Very difficult and very expensive.

But with Avocet's cross-assemblers, a single computer can develop software for virtually any microprocessor! Does that put us in a league with Bach? You decide.

The Well-Tempered Cross-Assembler

Development Tools That Work

Avocet cross-assemblers are fast, reliable and user-proven in over 3 years of actual use. Ask NASA, IBM, XEROX or the hundreds of other organizations that use them. Every time you see a new microprocessor-based product, there's a good chance it was developed with Avocet cross-assemblers.

Avocet cross-assemblers are easy to use. They run on any computer with CP/M* and process assembly language for the most popular microprocessor families.

5¼" disk formats available at no extra cost include Osborne, Xerox, H-P, IBM PC, Kaypro, North Star, Zenith, Televideo, Otrona, DEC.

Turn Your Computer Into A Complete Development System

Of course, there's more. Avocet has the tools you need from start to finish to enter, assemble and test your software and finally cast it in EPROM:

Text Editor VEDIT -- full-screen text editor by CompuView. Makes source code entry a snap. Full-screen text editing, plus TECO-like macro facility for repetitive tasks. Pre-configured for over 40 terminals and personal computers as well as in user-configurable form.

CP/M-80 version \$150
CP/M-86 or MDOS version \$195
(when ordered with any Avocet product)

EPROM Programmer -- Model 7128 EPROM Programmer by GTek programs most EPROMS without the need for personality modules. Self-contained power supply ... accepts ASCII commands and data from any computer through RS 232 serial interface. Cross-assembler hex object files can be down-loaded directly. Commands include verify and read, as well as partial programming.

PROM types supported: 2508, 2758, 2516, 2716, 2532, 2732, 2732A, 27C32, MCM8766, 2564, 2764, 27C64, 27128, 8748, 8741, 8749, 8742, 8751, 8755, plus Seeq and Xicor EEPROMS.

Avocet Cross-assembler	Target Microprocessor	CP/M-80 Version	•CP/M-86 IBM PC, MSDOS• Versions•
•XASMZ80	Z-80	\$200.00 each	\$250.00 each
•XASM85	8085		
XASM05	6805		
XASM09	6809		
XASM18	1802		
XASM48	8048/8041		
XASM51	8051		
XASM65	6502		
XASM68	6800/01		
XASMZ8	Z8		
XASMF8	F8/3870		
XASM400	COP400		
XASM75	NEC 7500	\$500.00	
Coming soon: XASM68K...68000			

(Upgrade kits will be available for new PROM types as they are introduced.)

Programmer \$389

Options include:

- Software Driver Package --
- enhanced features, no installation required.
- CP/M-80 Version \$ 75
- IBM PC Version \$ 95
- RS 232 Cable \$ 30
- 8748 family socket adaptor ... \$ 98
- 8751 family socket adaptor ... \$174
- 8755 family socket adaptor ... \$135

- **G7228 Programmer by GTek** -- baud
- to 2400 ... superfast, adaptive programming algorithms ... programs 2764 in one minute.

• Programmer \$499

- Ask us about Gang and PAL programmers.

- **HEXTRAN Universal HEX File Converter** -- Converts to and from Intel, Motorola, MOS Technology, Mostek, RCA, Fairchild, Tektronix, Texas Instruments and Binary formats.

- Converter, each version \$250

Call Us

If you're thinking about development systems, call us for some straight talk. If we don't have what you need, we'll help you find out who does. If you like, we'll even talk about Bach.

CALL TOLL FREE 1-800-448-8500

(In the U.S. except Alaska and Hawaii)

VISA and Mastercard accepted. All popular disc formats now available -- please specify. Prices do not include shipping and handling -- call for exact quotes. OEM INQUIRIES INVITED.

*Trademark of Digital Research

**Trademark of Microsoft



**AVOCET
SYSTEMS INC.™**

DEPT. 384-DD
804 SOUTH STATE STREET
DOVER, DELAWARE 19901
302-734-0151 TELEX 467210

by Robert Blum

Last month I ran a preliminary application note for CP/M Plus that reportedly would optimize the access time to any disk file — especially one that was small enough to fit into buffer memory and was accessed more than once within the same program. After installing the patches on my system, there appeared to be a significant increase in overall system performance, although at that time I had nothing more concrete to go on than an impression.

How Fast Is It?

To find out how much of a speed advantage (if any) was gained, I wrote a small benchmark program to be used in comparing the performance between an off-the-shelf distribution system and a modified one. The benchmark program I wrote had five distinct phases. The first four phases performed preliminary maintenance functions: make four files; write 16K of garbage data to each file; close the four files; and open the same four files that were just created.

The fifth and final phase sequentially read and reread the four files four times. If the changes were to be a success, this final phase of the program would produce the most dramatic evidence.

After completing the benchmark program, I copied the unaltered distribution files of my CP/M Plus system into a separate user area for use in link-editing the standard or unaltered CP/M system used in my benchmarks. My last step was to apply the patches outlined in the application note to one copy of my CP/M Plus system.

As shown in Table I (at right), four benchmarks were run, each with different combinations of features, etc. The first benchmark was run with two CP/M Plus systems that were completely stripped down and (as closely as I could make them) equal in performance to their V2.2 predecessor. As you can see, the runtime difference between the distribution and the modified system is small enough to be inconsequential.

The second and third benchmarks produced a healthy runtime reduction over the first one, but little difference was found between the two of them. This lack of improvement was a total surprise to me since I had expected the LRU buffering improvements to be more significant no matter what other options had been selected. But, as it appears, the LRU buffering logic is directly, or at least very closely, tied to the logic used for directory hashing.

Benchmark number four plainly points out how much of an improvement a plain vanilla CP/M Plus system can make in runtimes (not to mention the further reductions that can be experienced by tuning the system a little).

CP/M 2.2 BIOS Function: SELDSK

Billy Smith of Kentfield, California, writes:

“Here is a special treat for CP/M hackers. This tidbit just turned up as the root of a tricky little bug. I was implementing a public domain program called FILE.ASM at the time. Its function is to display all files on all drives and user areas matching the ambiguous file reference given in the command line. On my system it was stopping after completing drive A as if I had just done a cold boot and hadn’t referenced any other drives yet.

“Debugging revealed that there is a slight difference in the version 2.2 CP/M BIOS function SELDSK from all earlier versions. There is an additional sentence in the V2.2 manual, under SELDSK, that explains a little further: ‘The least significant bit of register E is zero if this is the first occurrence of the drive select since the last cold or warm start.’ There was no reference to register E having to be preset to any special value in earlier versions of the CP/M manual.

“My BIOS (Morrow) takes advantage of this information and does not do a for-

mal selection of a drive if the bit is non-zero. Instead it simply returns a value from a local variable that is assumed properly set during the first disk select. I suppose CP/M keeps this bit correct when calling SELDSK, but any program that does direct BIOS calls has the responsibility of managing this bit.

“One problem with this logic is that an application program has no idea if a drive has already been referenced or not and therefore must always set the bit to zero for its first reference even though this may be redundant to the BIOS. Since this public domain program had random non-zero data in register E, my BIOS was returning a bad value from SELDSK. In fact the bad value happened to be zero, which the program interpreted to mean non-existent drive and, of course, ended its operation.

“My simple fix was to clear register E before the call to SELDSK. Every call to SELDSK is now treated as though it were the first disk select of that drive and a proper value is returned.”

Application Note 2

Michael Carter of Garran, Australia, wrote several months ago to share a patch he had developed for reversing the meaning of the BACKSPACE and DELETE keys in CP/M V2.2. Michael’s note prompted me to dig through my files to see if DR had officially released anything

Run	Distribution	Modified
#1	76 seconds	74 seconds
#2	64 seconds	59 seconds
#3	62 seconds	61 seconds
#4	37 seconds	25 seconds

Legend

- #1 No features enabled, 1 Directory buffer and 1 Data buffer
- #2 Directory Hashing enabled, 1 Directory buffer and 1 Data buffer
- #3 No features enabled, 23 Directory buffers and 255 Data buffers
- #4 Directory Hashing enabled, 23 Directory buffers and 255 Data buffers

Table I.
Distribution vs. Modified CP/M Plus

on this subject. Sure enough, in February of 1982 an application note was published that thoroughly covers the subject.

CP/M® V2.2 Application Note 02, 2/20/82: Reversing the BACKSPACE and RUBOUT Key Functions and Making RUBOUT Identical to BACKSPACE

Copyright © 1982 by Digital Research. CP/M is a registered trademark of Digital Research. DDT and SID are trademarks of Digital Research. Compiled November 1982. Reprinted with permission of Digital Research. All information here is proprietary to Digital Research.

Applicable products and version numbers:
CP/M® V2.1 and V2.2

Program: BDOS

In the following code segment procedures, addresses given are hexadecimal offsets from the base of the CP/M system. The CCP is usually located at 980H but can be located at A00H if a two-sector boot is used.

You can assemble the patch for your size memory system. The cpmbase equals the BDOS entry point address at locations 6 and 7 in the base page of memory minus 806H. You must change this entry point address when you load DDT™ or SID™. Under DDT or SID, follow the jump at location 5 until an address is found with a least significant digit of 6. In the following example, the cpmbase would be E506H-806H or DD00H.

```
0005  JMP      CD00
CD00  JMP      D3A4
D3A4  XTHL
D3A5  SHLD    E452
D3A8  XTHL
D3A9  JMP      E506
```

Procedure to reverse the BACKSPACE and RUBOUT key functions:

Patch into the SYSGEN or MOVCPM image exactly as you would patch in a new version of your BIOS, using the DDT i command followed by the DDT r command. You can use the same offset as your custom BIOS and install the code found in Listing One (page 14).

Patch into the SYSGEN or MOVCPM image exactly as you would patch in a new version of your BIOS, using the DDT i command followed by the DDT r command. Use the same offset as your custom BIOS and install the code in Listing Three (page 14).

Or, you can install the above procedure directly into MOVCPM if you have MOVCPM.COM on your system disk. The patch is installed automatically in any size system that you build using MOVCPM. Make a back-up copy of MOVCPM.COM before using DDT to make the following changes:

```
A>ddt movcpm.com
DDT VERS 2.2
NEXT PC
```

```
2700 0100
-1141b
141B  MOV     A,B
141C  ORA     A
141D  JZ      09EF
1420  MOV     A,M
1421  DCR     B
...
-a141b
141B mvi a,8
141D jmp a07
1420 .
-g0
```

```
A> save 38 movcpml.com
```

Use the new program MOVCPM1.COM in place of MOVCPM.COM. The

RUBOUT and BACKSPACE key functions are identical in any CP/M system generated with MOVCPM1.COM.

Licensed users are granted the right to include these modifications in CP/M V2.2 software.

Or, you can install the above procedure directly into MOVCPM if you have MOVCPM.COM on your system disk. The patch is applied automatically to any size system that you build using MOVCPM. Make a back-up copy of MOVCPM.COM before using DDT to make the following changes:

```
A>ddt movcpm.com
DDT VERS 2.2
NEXT PC
```

"Q-PRO 4 blows dBASE II away"

We now complete complex applications in weeks instead of months."

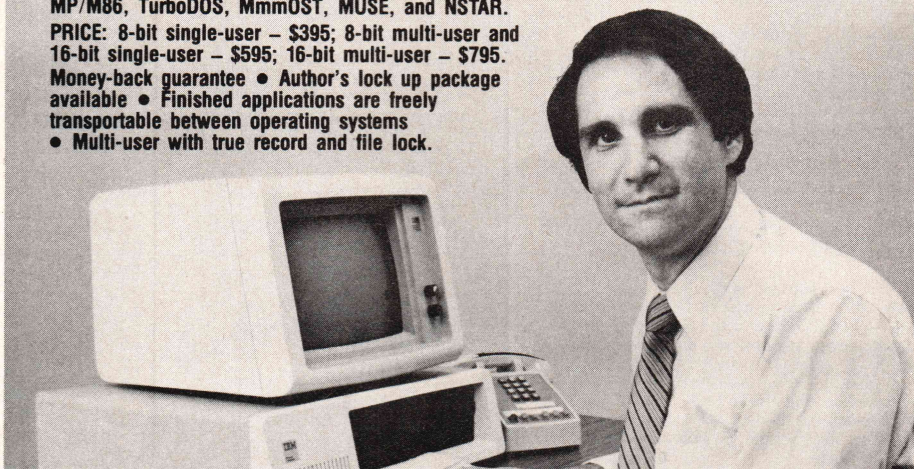
says Q-PRO4 user, Richard Pedrelli, President, Quantum Systems, Atlanta, GA

“As a dBASEII beta test site the past two years, we were reluctant to even try Q-PRO4. Now we write all our commercial applications in Q-PRO4. We find it to be an order of magnitude more powerful than dBASEII.

We used Q-PRO4's super efficient syntax to complete our Dental Management and Chiropractic Management Systems much faster. Superb error trap and help screen capabilities make our finished software products far more user friendly, too.

In my estimation, any application programmer still using outdated 3rd generation data base managers or worse, a 2nd generation language like BASIC, is ripping himself off."

Runs with PC DOS, MS-DOS, CP/M, MP/M, CP/M86, MP/M86, TurboDOS, MmmOST, MUSE, and NSTAR.
PRICE: 8-bit single-user - \$395; 8-bit multi-user and 16-bit single-user - \$595; 16-bit multi-user - \$795.
Money-back guarantee • Author's lock up package available • Finished applications are freely transportable between operating systems
• Multi-user with true record and file lock.



For Q-PRO 4 demonstration, go to nearest MicroAge store or other fine dealer.

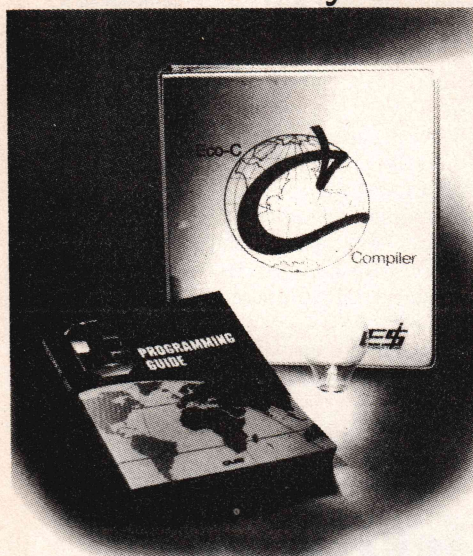
quic·n·easi products inc.

136 Granite Hill Court, Langhorne, PA 19047 (215) 968-5966 Telex 291-765

CP/M, MP/M, CP/M86, and MP/M86 are trademarks of Digital Research. TurboDOS, MmmOST, MUSE, NSTAR, MS-DOS and PC DOS are trademarks of Software 2000. TaiVideo Systems, O.S.M., Molecular, Microsoft and IBM, respectively.

Circle no. 48 on reader service card.

We make C easy ...



and work!

Eco-C compiler... we've got it all.

Whether you're a seasoned professional or just getting started in C, the Ecosoft C compiler has everything you'll ever need.

COMPLETENESS:

Our Eco-C compiler is a complete implementation of C and supports all operators and data types (including long, float and double).

EFFICIENCY:

The compiler generates extremely efficient Z80 code using Zilog's mnemonics. On the benchmarks tested, typically we finished either first or second using substantially less generated code.

PORTABILITY:

The ECO-C library contains over 100 functions that are UNIX V7 compatible, and includes a complete transcendental package. Programs developed with the Eco-C compiler can be moved to virtually any system with little or no change.

EASE OF USE:

The Eco-C compiler includes Microsoft's MACRO 80 assembler, linker, library manager and supporting documentation. The assembler (M80) generates industry-standard REL file output. The linker (L80) is fast and uses only the functions you request in the program. Program development is a snap.

The user's manual is clear, concise and full of useful information. For those of you just getting started with C, we also include a copy of the **C Programming Guide** (Que). This B. Dalton Best Seller has been adopted by a number of leading universities around the country and is included with each compiler. The book is designed to help you learn C from the ground up. We ought to know... we wrote the book.

We've made the compiler easy to work with for the professional and beginner alike. Most error messages, for example, tell you in English (not just a number) the line number and character position of the error, what was expected and a page reference to the **Guide** to consult for help if you need it.

PRICE:

We saved the best for last; we've cut the price by \$100.00. Now you can buy the Eco-C compiler for only \$250.00 (MACRO 80 and the book alone are worth \$218.00!). Shop around and we think you agree that the Eco-C compiler is the best value available.

The Eco-C compiler requires a Z80 CPU, CP/M, 54K of free memory and about 240K of disk space (one or two drives). An IBM-PC version will be available in the first quarter of 84. To order your Eco-C compiler, call or write.



Ecosoft Inc.
P.O. Box 68602
Indianapolis, IN 46268
(317) 255-6476



TRADEMARKS:
Eco-C (Ecosoft), MACRO 80 (Microsoft), CP/M (Digital Research)

Circle no. 21 on reader service card.

```
2700 0100
-11402
1402 CPI 08
1404 JNZ 0A16
1407 MOV A,B
1408 ORA A
1409 JZ 09EF
140C DCR B
140D LDA 0B0C
1410 STA 0B0A
1413 JMP 0A70
1416 CPI 7F
1418 JNZ 0A26
-s1403
1403 08 7f
1404 C2 .
-s1417
1417 7f 8
1418 C2 .
```

-g0

A>save 38 movcpml.com

Use the new program MOVCPM1.COM in place of MOVCPM.COM. The BACKSPACE and RUBOUT key functions are reversed for any CP/M system generated with MOVCPM1.COM.

Procedure to make RUBOUT identical to BACKSPACE:

Before you install this patch, the code at cpmbase + 0A1Bh should read as shown in Listing Two (below). **DDJ**

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 191.

CP/M Exchange Listing (Text begins on page 12)

Listing One

```
cpmbase equ      ?      ;subtract 806h from address
                        at location 6
                        org cpmbase + 0A02h
                        cpi 7fh ;was cpi 08h
                        org cpmbase + 0A16h
                        cpi 08h ;was cpi 7fh
```

End Listing One

Listing Two

```
mov a,b
ora a
jz cpmbase + 09EFh
mov a,m
dcr b
dcx h
jmp cpmbase + 0AA9h
```

End Listing Two

Listing Three

```
cpmbase equ      ?
;
;
;      org cpmbase + 0A1Bh
;
mvi a,8h
jmp cpmbase +0A07h
end
```

End Listing

THE INTERNATIONAL VIDEO GAME OF THE YEAR COMPETITION HERALDS THE "NEW AGE" IN VIDEO/COMPUTER GAMES

\$175,000 TO BE WON

**AND THAT'S
JUST FOR
STARTERS!**



Create a brilliant, new video game and you could be on your way to becoming a millionaire. This fantastic competition, organised by I.R.P. (The International Register of Independent Computer Programmers Ltd) and the famous Mark McCormack International Management Group, offers programmers and inventors the opportunity of a life time. There are huge, immediate cash prizes and the on-going revenue of 10% of the sales of all games to distributors throughout the world, plus the chance to appear on an international TV show. Your skill and imagination could bring you fame and fortune!

\$100,000 FIRST PRIZE! PLUS FIVE \$15,000 RUNNER-UP PRIZES!

Devise a totally original new video game in one of these categories: SPORTS, SIMULATORS, ARCADE, STRATEGY, ADVENTURE/FANTASY or a special section which covers programmes that are not necessarily games but have outstanding Educational or Entertainment merit. We'll also be announcing a number of 'MERIT' awards which will be entitled to carry the message 'An International Video Game of the Year MERIT AWARD' on their retail packaging. It's a great challenge. And the rewards, both financially and in terms of prestige, are tremendous. This is the most exciting competition ever for creative computer and video enthusiasts.

YOU'RE A TV STAR TOO! All six winning games and their inventors will be featured on an internationally distributed, spectacular TV special. That's going to make your name!

HOW TO ENTER

Just send in your game, or games, programmed on cassette for any popular home computer. Use the coupon, today, and we'll send you all the facts you need.

**CLOSING DATE FOR ENTRIES IS
31st MAY 1984**

To: IRP Limited, Pinewood
Film Studios, Iwer, Bucks,
England.

Name

Address

.....

.....

.....

DJ1

RSA: A Public Key Cryptography System, Part I

Cryptography has been in existence since the beginning of written history. Most of these ciphers and codes were developed for military applications. One of the more well-known systems, designed by Julius Caesar, is a simple letter transposition cipher that is easily broken today. Since that time, cryptography has matured; some current systems are estimated to be unbreakable, even using state-of-the-art technology. Some claim that cryptographers would have to spend millions of years using today's fastest computers to break some of these ciphers.

Key Systems

In the late 1970s, IBM introduced a single key system that was later made into a standard by the National Bureau of Standards. The Data Encryption Standard (DES) is a 56-bit key system that has been committed to silicon by several semiconductor vendors. The cipher is fast and easy to integrate into LSI circuits. During and since its standardization, controversy has raged over the security of the DES. The primary concern is over the short length of the key. Many also speculate that since the government, which has overriding national security concerns, standardized the cipher, they may have a means of breaking it. However, many groups use the DES, including financial institutions, local network manufacturers, and others.

The Public Key System (PKS) was first proposed by W. Diffie and M. E. Hellman in 1976.¹ Their paper described a dual key system with keys generated from large prime numbers (100+ decimal digits). In 1979, Hellman published an article in *Scientific American* describing the mathematics of the PKS.² He also offered a monograph that described the system in greater detail, but within a year of the monograph offering, it was withdrawn from public distribution. Specula-

"In this two-part series of articles, we will discuss the Rivest-Shamir-Adleman (RSA) PKS and show how to implement it on a microcomputer."

tion was that the government (NSA and DoD) had forced its withdrawal because it jeopardized the national security. Unlike the DES cipher, the PKS has not been committed to silicon because it is much more complex and requires considerably more mathematics and computation time, as we will soon see. Currently, investigators continue ongoing research efforts to generate efficient integrated circuit implementations of the PKS.

In 1978, R. L. Rivest, A. Shamir, and L. Adleman showed how the PKS could be implemented and proposed a means of providing digital signatures to the messages.³ The addition of a signature to a message allows a sender to sign his message so that a receiver can be sure that the message originated with the sender. The February 1983 issue of *IEEE Computer* had a series of articles discussing ways to compromise digital signatures and how they could be made more secure.^{4, 5, 6, 7} Some of the articles examined several different ciphers, their basic implementations, and their limitations.

In this two-part series of articles, we will discuss the Rivest-Shamir-Adleman (RSA) PKS and show how to implement it on a microcomputer. The first part will discuss RATFOR (the implementation language) and the mathematical core of the RSA system, including modulo arithmetic, multiple-precision arithmetic, and "Russian Peasant" exponentiation. In the second part, we will describe the generation of the keys (public and private) and the encryption/decryption system; we will also take a brief look at digital signatures.

RATFOR

RATFOR (RATional FORtran) is a preprocessor to Fortran; i.e., the output of the RATFOR precompiler is Fortran. The language is described in the book *Software Tools*.⁸ RATFOR adds several structured constructs to Fortran while allowing use of standard Fortran statements, in-line comments, free-form input, multiple statements in a line, definitions, inclusion of other files, logical conditions

similar to BASIC or C, a standard library, and so on. In a lot of ways, it is similar to the C programming language, and those familiar with C should have little trouble transporting RATFOR source to C source. For that matter, those familiar with Pascal should be able to transport the software easily.

Those who want to use RATFOR can obtain a public domain version from the CP/M Users Group (CPMUG), operated by LifeBoat Associates, or from one of the RCPM bulletin boards that has it available for downloading. Unfortunately, a library is unavailable with the public domain version; however, using the *Software Tools* text, you can generate your own library of functions or simply pull out the appropriate functions from the precompiler.

Now let's review some of the rudiments of the language. The structured constructs that RATFOR adds to Fortran include:

```
repeat
{
:
statement n
:
}
until (condition)
while (condition)
{
:
statement n
:
}
for (init. statement; condition; loop
statement)
{
:
statement n
:
}
do n=j,k # Note the lack of a ter-
# minating label number
{
:
statement n
```

by C. E. Burton

C. E. Burton, 1720 S. Deframe Court,
Denver, CO 80228.

Copyright © 1983 by C. E. Burton. All rights reserved. Permission is granted for personal, noncommercial use only.


```

    }
if (condition)
{
    :
    statement x
    :
}
else if (condition)
{
    :
    statement y
    :
}
else
{
    :
    statement z
    :
}

```

RATFOR uses # as an in-line comment indicator and ignores anything after its occurrence to the end of the line. The { and } symbols are used as Begin and End designators, respectively. Within the iterative constructs, you can use the special words *break* and *next*; *break* causes an exit from the innermost loop, while *next* causes the innermost loop to continue execution at its "condition" statement. Because of the form of the "if" statement that RATFOR uses, you cannot use a Fortran arithmetic IF within RATFOR. For example,

```

if (condition) less-label, equal-label,
greater-label

```

is a "no-no."

The "condition" statements can contain any of the following logical operators:

```

== Equals
!= Not Equal
< Less Than
<= Less Than or Equal
> Greater Than
>= Greater Than or Equal
! Not
& And
| Or

```

The last three logical operators, !, &, and |, can also be used as logical bitwise operators on logical or integer variables.

Two other special words that are available are *define* and *include*. *define* allows you to make substitutions of strings within the source file. Beginning where it occurs within the source, it substitutes the right string for the left string throughout the text. An example follows:

```

define(Yes,1) # substitute "1" for
               # "YES" before processing

```

include allows you to include another file at the point that the statement occurs.

```

if ( ( (variable1 == variable2) & (variable3 != variable4) ) |
      (variable5 < variable6) )

call subroutine (variable1, variable2, variable3, variable4,
                variable5, variable6)

longline=variable1*(cos(variable2)/tan(variable3)+variable4*
                variable5)-variable6

```

Figure 1.

For example:

```

include RATDEF # reads and com-
               # piles the file RATDEF.RAT

```

Multiple statements per line are handled using semicolons, as shown below:

```

statement 1; statement 2; statement 3

```

Statements can also be extended across multiple lines. The "condition" statements are automatically extended; statements with comma separators are extended if the comma ends the line; but other statements must use an underscore (_) to extend across a line boundary. Figure 1 (above) illustrates these three extensions. The precompiler limits the line to 72 characters, generates a continuation character in column 6 of the next physical line, and continues the source in this

fashion until the logical line ends. To get an idea of how RATFOR handles these items, consult Listing One (page 22) and Listing Two (page 24) for examples of the RATFOR source and Listing Three (page 24) for the Fortran source generated by the precompiler.

Modulo Arithmetic

At the heart of a PKS cipher is the ability to perform multiple-precision, nonnegative, integer arithmetic. As was implied earlier, the PKS cipher requires the use of addition, subtraction, multiplication, division, and exponentiation of large numbers (100 to 250 decimal digits). Most computers, including microcomputers, are limited to arithmetic on numbers

WHEN IT COMES TO THE TWO BEST WORD PROCESSORS

There's only one main difference: price.* Because NewWord, with its built-in Merge Print, is keystroke, command and file compatible with MicroPro's WordStar®/MailMerge®.

NewWord also offers advanced design features like Unerase deleted text, automatically changing ruler lines, multiple line Headers and Footers, and on-screen display of boldfacing and underlining. NewWord is demonstrably superior on your dot matrix printer, supporting microjustification and variable line heights/character widths.

Make a comparison. NewWord is lighter only on your checkbook.

*Manufacturer's suggested retail

Call us today, toll-free **800-832-2244**
(In California, call 800-732-2311)



ROCKY MOUNTAIN SOFTWARE SYSTEMS
1280-C NEWELL AVE., SUITE 1052; WALNUT CREEK, CA 94596

WordStar and MailMerge are registered trademarks of MicroPro International, Inc. NewWord and Newstar are trademarks of Newstar Software, Inc.

having 3 to 10 decimal digits (usually in binary form). To compound the problem, performing the RSA algorithms requires the use of modulo arithmetic. In some respects, modulo arithmetic is a lifesaver in that it limits the size of the numbers that are generated; however, it also complicates things because additional processing must be done on the results of arithmetic operations.

While modulo arithmetic is similar to the arithmetic that we learned in school, it has some differences. Thus, we should look at a few of its properties. A "modulus" is essentially the base of the number system that is being employed, and the mod operator leaves the "residue" of numbers. We could define the operation $(A \bmod B)$ for nonnegative integers (in which we are currently interested) as:

$$A \bmod B = A - (B * \text{int}(A / B)) = \text{remainder}(A / B)$$

Therefore, a number system with a base of 10 (decimal) can only contain the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The following example uses hex (base 16):

$$75 \bmod 16 = 75 - 16 * \text{int}(75/16) = 75 - 16 * 4 = 75 - 64 = 11 < 16$$

Another interesting property of modulo arithmetic is:

$$(A @ B) \bmod C = ((A \bmod C) @ (B \bmod C)) \bmod C$$

where @ can represent +, -, or *. Exponentiation is similar:

$$(A ** B) \bmod C = ((A \bmod C) ** B) \bmod C$$

These operations are also commutative; i.e., the order of performance is not important, as illustrated in Figure 2a (at right). Figure 2b (below) shows a couple of examples to make sure we have it.

Now you can understand why using modulo arithmetic in raising large numbers to large powers can be a lifesaver. However, it does add considerably more operations (one division to get the remainder after each operation).

Multiple-Precision Arithmetic

In this section, we will cover the basic operations of addition, subtraction, multiplication, and division of nonnegative integers. I should point out that there are a few programs capable of performing multiple-precision arithmetic on very large numbers, e.g., MUMATH. These programs undoubtedly use similar algorithms to perform their operations, and I am sure that they could be put to use effectively in implementing the RSA algorithm. Our current objective, however, is to show how to do it.

The general opinion is that D.E. Knuth has written the "bible" of computer algorithms, and his work is the basic source of most of what I will subsequently pre-

sent in this set of articles. Many of the extensions to what he presents in the text of his books can be found in the problems at the end of each section and in the subsequent answers at the end of each book.⁹ Another paper dealing with multiple-precision arithmetic that might be helpful here was written by M. Zimmerman.¹⁰ My implementation of the multiple-precision arithmetic (MPARITH.RAT) can be found in Listing Four (page 26).

Looking at the basic arithmetic operations, we can say that:

1. Addition of two N-digit numbers produces a result that has at most (N+1) digits.
2. Subtraction of two N-digit numbers produces a result that has at most N digits.
3. Multiplication of an N-digit number by an M-digit number produces a result that has at most (N+M) digits.
4. Division of an (N+M)-digit number by an N-digit number produces a quotient of at most (M+1) digits and a remainder of at most N digits.

These statements hold true no matter what radix (number base) system is used. In my particular case, I have used a radix of 128, where each byte holds a

number between 0 and 127, because the Fortran that I used (Microsoft F80) does two's-complement arithmetic on byte variables. A *define* statement near the beginning of the routines to be presented in Part II sets BYTEMODULUS to 128. The MPARITH routines shown in Listing Four, however, are not limited to this value. They allow any byte modulus (radix) between 2 and 128, and the value is passed to the subroutine by the parameter MODULO.

Knuth, in his set of books dealing with computer programming, presents both algorithms and the "MIX" assembly language programs that implement the algorithms. I have implemented a RATFOR (Fortran) version of the algorithms. No doubt, an assembly version of the algorithms written in Microsoft's M80 would run considerably faster, but I was more interested in transportability and ease of understanding than in speed. (A basic premise of programming is to get it working first then to modify things to optimize performance only if it is necessary. Another premise is that if it ain't broke, don't fix it!)

I will present Knuth's algorithms here without going into detail on my routines.* Since RATFOR allows in-line comments, the code should include most of the re-

*Donald E. Knuth, *The Art of Computer Programming, Vol. 2*, ©1981, Addison-Wesley, Reading, Massachusetts. Pgs. 250, 252, 253, 254, 257, 258 and 442 (to include Algorithms "A", "S", "M", "D", "A"). Reprinted with permission.

$$\begin{aligned} (A @ B) \bmod C &= (B @ A) \bmod C \\ (A ** (B * C)) \bmod D &= ((A ** B) ** C) \bmod D \\ &= (((A ** B) \bmod D) ** C) \bmod D \\ &\quad \text{— or —} \\ &= ((A ** C) ** B) \bmod D \\ &= (((A ** C) \bmod D) ** B) \bmod D \end{aligned}$$

Figure 2a.

$$\begin{aligned} (5 * (6+2) - 4) \bmod 7 &= 36 \bmod 7 = 1 \\ &= (5 * (8 \bmod 7) - 4) \bmod 7 = (5 - 4) \bmod 7 = 1 \\ (3 ** (4 * 2)) \bmod 7 &= 6561 \bmod 7 = 2 \\ &= (((3 ** 4) \bmod 7) ** 2) \bmod 7 \\ &= ((81 \bmod 7) ** 2) \bmod 7 = (4 ** 2) \bmod 7 \\ &\quad \text{— or —} \\ &= 16 \bmod 7 = 2 \\ &= (((3 ** 2) \bmod 7) ** 4) \bmod 7 \\ &= ((9 \bmod 7) ** 4) \bmod 7 = (2 ** 4) \bmod 7 \\ &= 16 \bmod 7 = 2 \end{aligned}$$

Figure 2b.

quired documentation, especially when it is compared to the original algorithms. A transcription of these algorithms follows with some minor rewording. These algorithms are presented for those of you who do not have access to Knuth's second volume.

Addition Algorithm

ALGORITHM A (addition of non-negative integers). Given nonnegative N -place integers (u_1, u_2, \dots, u_n) radix b and (v_1, v_2, \dots, v_n) radix b , this algorithm forms their sum $(w_0, w_1, w_2, \dots, w_n)$ radix b . (Here w_0 is the carry, and it will always be 0 or 1.)

- A1. [Initialize] Set $j=n$, $k=0$. (The variable j will run through the various digit positions, and the variable k keeps track of carries at each step.)
- A2. [Add digits] Set $w_j = (u_j + v_j + k) \bmod b$ and $k = \text{floor}((u_j + v_j + k) / b)$. (In other words, k is set to 1 or 0, depending on whether a carry occurs or not, i.e., whether $u_j + v_j + k \geq b$ or not. At most one carry is possible during the two additions, since we always have $u_j + v_j + k \leq 2 * (b - 1) + 1 < 2 * b$, by induction on the computation.)
- A3. [Loop on j] $j=j-1$. If $j > 0$, then go to A2; else set $w_0=k$ and terminate.

Note: $\text{floor}(x)$ is the greatest integer less than or equal to x , also known as $\text{int}(x)$.

Subtraction Algorithm

ALGORITHM S (subtraction of non-negative integers). Given nonnegative N -place integers (u_1, u_2, \dots, u_n) radix b and (v_1, v_2, \dots, v_n) radix b , this algorithm forms their nonnegative difference (w_1, w_2, \dots, w_n) radix b .

- S1. [Initialize] Set $j=n$, $k=0$.
- S2. [Subtract digits] Set $w_j = (u_j - v_j + k) \bmod b$ and $k = \text{floor}((u_j - v_j + k) / b)$. (In other words, k is set to -1 or 0, depending on whether a borrow occurs or not, i.e., whether $u_j - v_j + k < 0$ or not. In the calculation of w_j , note that we must have $-b \leq u_j - v_j + k \leq b$ or $0 \leq u_j - v_j + k + b \leq 2 * b$.)
- S3. [Loop on j] $j=j-1$. If $j > 0$, then go to S2; else terminate. (When the algorithm terminates, k should be 0; the condition $k=-1$ will occur if and only if $V > U$, and this is contrary to the assumptions.)

Multiplication Algorithm

ALGORITHM M (multiplication of nonnegative integers). Given nonnegative integers (u_1, u_2, \dots, u_n) radix b and (v_1, v_2, \dots, v_m) radix b , this algorithm forms the product (w_1, w_2, \dots, w_l) radix b , $l=m+n$. (The conventional pencil-and-paper method is based on forming the partial products $(u_1, u_2, \dots, u_n) * v_j$ first, for $1 \leq j \leq m$, and then adding

these products together with appropriate scale factors; in a computer it is best to do the addition concurrently with the multiplication, per this algorithm.)

- M1. [Initialize] Set $w_j=0$, $m+1 \leq j \leq m+n$. Set $j=m$. (If w_j above were not set to zero in this step, it turns out that the steps below would set $W = U * V + (w_i, \dots, w_j)$, $i=m+1$ and $j=m+n$. This more general operation is sometimes useful.)
- M2. [Zero multiplier?] If $v_j = 0$, set $w_j=0$ and go to Step M6. (This test saves a good deal of time if there is a reasonable chance that v_j is zero, but otherwise it may be omitted without affecting the validity of the algorithm.)
- M3. [Initialize i] Set $i=n$ and $k=0$.
- M4. [Multiply and add] Set $t = u_i * v_j + (w_r + k)$, $r=i+j$; then set $w_r = (t \bmod b)$ and $k = \text{floor}(t / b)$. (Here the carry k will always be in the range $0 \leq k < b$.)
- M5. [Loop on i] $i=i-1$. Now if $i > 0$, go back to Step M4; otherwise set $w_j=k$.
- M6. [Loop on j] $j=j-1$. Now if $j > 0$, go back to Step M2; otherwise the algorithm terminates.

Division Algorithm

ALGORITHM D (division of nonnegative integers). Given nonnegative integers

(u_1, u_2, \dots, u_l) radix b , $l=m+n$, and (v_1, v_2, \dots, v_n) radix b , where $v_1 \neq 0$ and $n > 1$, we form the quotient $\text{floor}(U/V) = (q_1, q_2, \dots, q_m)$ radix b and the remainder $U \bmod V = (r_1, r_2, \dots, r_n)$ radix b .

- D1. [Normalize] Set $d = \text{floor}(b / (v_1 + 1))$. Then set (u_0, u_1, \dots, u_l) radix b , $l=m+n$, equal to (v_1, v_2, \dots, v_n) radix b times d . (Note the introduction of a new digit position u_0 at the left of u_1 ; if $d=1$, all we need to do in this step is to set $u_0=0$. On a binary computer it may be preferable to choose d to be a power of two instead of using the value suggested here; any value of d that results in $v_1 \geq \text{floor}(b/2)$ will suffice.)
- D2. [Initialize j] Set $j=0$. (The loop on j , Steps D2 through D7, will be essentially a division of (u_j, \dots, u_r) radix b , $r=j+n$, by (v_1, \dots, v_n) radix b to get a single quotient digit q_j .)
- D3. [Calculate q'] If $u_j=v_1$, set $q'=b-1$; otherwise set $q' = \text{floor}((u_j * b + u_r) / v_1)$, $r=j+1$. Now test if $v_2 * q' > (u_j * b + u_r - q' * v_1) * b + u_s$, $r=j+1$ and $s=j+2$; if so, decrease q' by 1 and repeat this test. (The latter test determines at high speed most of the cases in which the trial value q' is one too large, and it eliminates all cases where

No more lost edit changes!

COMPARE II

High performance differential text analyzer!

```

graph LR
    A[New text file] --> B[COMPARE II]
    C[Old (reference) text file] --> B
    B --> D[Differences to file]
    B --> E[Differences to terminal]
    B --> F[Differences to printer]
  
```

Writers! Researchers! Lawyers! Engineers! Programmers!
Use highly rated COMPARE II. Cut text analysis from hours to minutes!

- ▶ PC-DOS, CP/M-86 or CP/M 2.2
- ▶ Scans by word or by line
- ▶ Fast New Algorithm, No file restrictions
- ▶ You can customize for word processor, printer width, file defaults, specific work flow, computer languages, different highlighting techniques
- ▶ Clear commands, Numerous formatting options
- ▶ Can generate new document with change bars

Specify When Ordering: Operating System, Computer Type and Disk Format. Free brochure and *nearly* free demo disk available.

COMPARE II

Demo Disk (credits to purchase)..... \$12.95

SOLUTION TECHNOLOGY, INC.

PC-DOS and CP/M are trademarks of IBM and Digital Research respectively.

\$145.00

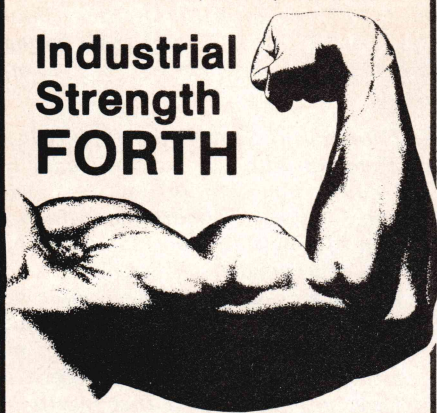
"We Deliver Productivity"

1499 Palmetto Park Road
 Suite 218
 Boca Raton, FL 33432
 305/368-6228

Check or COD. Florida residents add 5% sales tax.
 Dealer and Distributor inquiries welcome.

**Multiuser/Multitasking
for 8080, Z80, 8086**

Industrial Strength FORTH



TaskFORTH™

The First
Professional Quality
Full Feature FORTH
System at a micro price*

LOADS OF TIME SAVING PROFESSIONAL FEATURES:

- ☆ Unlimited number of tasks
- ☆ Multiple thread dictionary, superfast compilation
- ☆ Novice Programmer Protection Package™
- ☆ Diagnostic tools, quick and simple debugging
- ☆ Starting FORTH, FORTH-79, FORTH-83 compatible
- ☆ Screen and serial editor, easy program generation
- ☆ Hierarchical file system with data base management

* Starter package \$250. Full package \$395. Single user and commercial licenses available.

If you are an experienced FORTH programmer, this is the one you have been waiting for! If you are a beginning FORTH programmer, this will get you started right, and quickly too!

**Available on 8 inch disk
under CP/M 2.2 or greater
also
various 5¼" formats
and other operating systems**

**FULLY WARRANTIED,
DOCUMENTED AND
SUPPORTED**



DEALER
INQUIRES
INVITED



Shaw Laboratories, Ltd.
24301 Southland Drive, #216
Hayward, California 94545
(415) 276-5953

q' is two too large.)

D4. [Multiply and subtract] Replace (uj, ..., ur) radix b, $r=j+n$, by (uj, ..., ur) radix b minus q' times (v1, ..., vn) radix b. This step (analogous to Steps M3, M4, and M5 above) consists of a simple multiplication by a one-place number, combined with a subtraction. The digits (uj, ..., ur) radix b should be kept positive; if the result of this step is actually negative, (uj, ..., ur) radix b should be left as the true value plus $b^{**}(n+1)$ (i.e., as the b's complement of the true value) and a borrow to the left should be remembered.

D5. [Test remainder] Set $qj=q'$. If the result of Step D4 was negative, go to Step D6; otherwise go to Step D7.

D6. [Add back] (The probability that this step will be necessary is very small, on the order of only $2/b$; test data that activate this step should therefore be specifically contrived when debugging.) $qj=qj-1$, and add (0, v1, ..., vn) radix b to (uj, ..., ur) radix b, $r=j+n$. (A carry will occur to the left of uj, and it should be ignored since it cancels with the borrow that occurred in D4.)

D7. [Loop on j] $j=j+1$. Now if $j \leq m$, go back to D3.

D8. [Un-normalize] Now ($q0, q1, \dots, qm$) radix b is the desired quotient, and the desired remainder may be obtained by dividing (ur, ..., us) radix b, $r=m+1$ and $s=m+n$, by d.

As you can see, the implementation of these algorithms in Listing Four follows very closely, with the exception of some differences or additions that were required to make things work properly. A few areas are also identified in the listing where additions to the algorithms can be implemented. Obviously, these algorithms are very computation-intensive, and they form the root of the rest of the software. If we were to expend effort to optimize these routines, a real savings in execution time could undoubtedly be realized. A possibility would be to change the number arrays from byte-types to integer-types and to increase the arithmetic modulus MODULO. However, this change would create additional complications in handling text files because character packing would now have to be done. Another method to speed up the operations would be to use an arithmetic coprocessor, e.g., an AMD 9511 or an Intel 8087. Speed increases of up to an order of magnitude could potentially be realized.

We are now ready to discuss the final mathematical algorithm in this part of the article, the "Russian Peasant" algorithm. It will use the multiple-precision arithmetic algorithms just presented.

Russian Peasant Exponentiation

This algorithm was given its name by nineteenth-century visitors to Russia who found the technique in wide use there. However, the algorithm appeared in the fifteenth century and is based on a multiplication technique developed by the Egyptians at least 3300 years earlier. In certain aspects, the algorithm is similar to a binary search. The algorithm, as stated by Knuth, follows.

Russian Peasant Algorithm

ALGORITHM A (right-to-left binary method for exponentiation). This algorithm evaluates $X^{**}N$, where N is a positive integer.

A1. [Initialize] Set $n=N, y=1, z=X$.

A2. [Halve n] (At this point $X^{**}N = y^{**}z^{**}n$.) Determine whether n is odd or even. Set $n=\text{floor}(n/2)$. If n is even, go to Step A5.

A3. [Multiply y by z] Set $y = y * z$.

A4. [Check n for 0] If $n = 0$, the algorithm terminates with y as the answer.

A5. [Square z] Set $z = z * z$, and go to Step A2.

It should be pointed out that a dual algorithm for multiplication ($y = X * N$) can be achieved by replacing $y=1$ with $y=0$ in Step A1; by replacing Step A3 with an addition, i.e., $y = y + z$; and by replacing Step A5 with a doubling instead of a squaring, i.e., $z = z + z$. J. Nyberg has presented a BASIC program to perform this algorithm.¹¹ This method of exponentiation is usually suitable only for large exponents, N, since multiple-precision multiplication is more efficient for small and moderate values of N. Some faster methods are available. One involves the factoring of N into its prime factors. Another method uses a technique called "power trees." These methods are explained in more detail in Knuth's book, but these techniques are more complex than the Russian Peasant method.

The Russian Peasant algorithm had to be broadened to perform $Y = (X^{**}N) \text{ mod } M$. The modulo operation limits the resultant answer to $0 \leq Y < M$. We shall see in the next part of the article that this form of the algorithm is required. The modification to the algorithm is as follows:

A5. [Square z] Set $z = z * z$.

A6. [Limit z] Set $z = (z \text{ mod } M)$ and go to A2.

The implementation of the expanded algorithm (RPEASANT.RAT) is found in Listing Five (page 38). The listing contains three subroutines:

RPEXP – The modified Russian Peasant algorithm.

PRDMOD – The routine that performs $(Z * Z) \text{ mod } M$. It uses the multiple-

precision multiplication and division routines given in Listing Four.

MAKBIN — This routine tries to make the halving of the exponent, N, somewhat more efficient. The byte modulus, MODULO, does not have to be $2^{**}K$, $0 < K \leq 7$; thus, N cannot be strictly scanned on a bit-by-bit basis to perform the even/odd check and halving. So we must find a base modulus so that $BASMOD = 2^{**}\text{ceiling}(\log_2(\text{MODULO}))$, and use it to strip off bytes that can be scanned on a bit-by-bit basis in the even/odd check. The operation becomes:

1. Initially, set $N_x = N$ (first pass through)
2. $TEMP = \text{remainder}(N_x / \text{BASMOD})$
3. $N_x = \text{floor}(N_x / \text{BASMOD}) = \text{quotient}(N_x / \text{BASMOD})$

where TEMP is used by RPEXP to do Step A2 and the new N_x is retained for the next pass until $N_x = 0$. If $\text{MODULO} = 2^{**}K$, TEMP just returns the current least-significant byte of N_x without doing the division. Therefore, it is most efficient to make $\text{MODULO} = \text{member}\{2, 4, 8, 16, 32, 64, 128\}$.

As you can see in the multiple-precision routines and in the Russian Peasant routine, a considerable amount of pointer manipulation occurs. These routines probably could be implemented more efficiently in C or in Pascal, where pointer manipulation is more easily attained. I do not claim that these implementations are optimal; however, they do work. If you feel the need to "optimize" the code, dig in!

Summary

We are now at the end of the first part of the article. We have covered several areas: RATFOR, modulo arithmetic, multiple-precision arithmetic, and Russian Peasant exponentiation. If you implement these algorithms on your system, you should be able to check the addition and subtraction algorithms fairly easily. To check out the multiplication, division, and exponentiation algorithms, you will need a calculator with a large digit accuracy (≥ 10 is desirable) or a copy of MUMATH or TK!SOLVER. Try this problem:

```
D = (A ** B) mod C
using
A = 9182736450
B = 1928374605
C = 12345678907
Byte Modulo = 100
to obtain
D = 10447988731
```

In the second and final part, we will look at the RSA-PKS encryption/decryp-

tion implementation. This will cover three additional areas: the generation and testing of large prime numbers, the generation of the public/private keys, and the encryption/decryption of text files with a brief look at digital signatures. See you on the second pass.

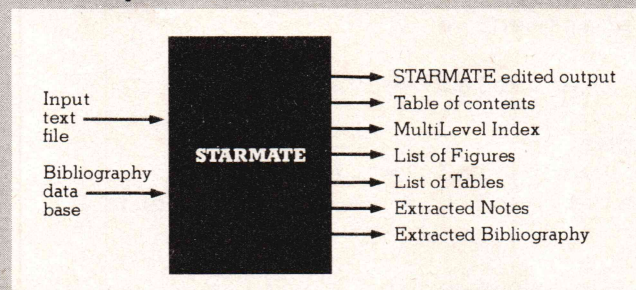
References

- ¹ W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Trans Info Theory*, Vol. IT-22, No. 6, pp. 644-654.
- ² M.E. Hellman, "The Mathematics of Public Key Cryptography," *Sci Amer*, February 1979, Vol. 241, No. 2, pp. 146-157.
- ³ R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Comm ACM*, February 1978, Vol. 21, No. 2, pp. 120-126.
- ⁴ S.G. Akl, "Digital Signatures: A Tutorial Survey," *Computer*, February 1983, Vol. 16, No. 2, pp. 15-24.
- ⁵ D.W. Davies, "Applying the RSA Digital Signature to Electronic Mail," *Computer*, February 1983, Vol. 16, No. 2, pp. 55-62.
- ⁶ R. DeMillo and M. Merritt, "Protocols for Data Security," *Computer*, February 1983, Vol. 16, No. 2, pp. 39-50.
- ⁷ D.E. Denning, "Protecting Public Keys & Signature Keys," *Computer*, February 1983, Vol. 16, No. 2, pp. 27-35.
- ⁸ B.W. Kernighan and P.J. Plauger, *Software Tools*, Reading, MA: Addison-Wesley, 338 pp.
- ⁹ D.E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Reading, MA: Addison-Wesley, pp. 250-268, 374-380, 386-389, 442-443.

Get more than WordIndex for less \$!

STARMATE

Users get high speed document finishing for WordStar with STARMATE under PC-DOS, CP/M-86, or CP/M 2.2.



Writers! Researchers! Engineers! Cut document makeup time from hours to minutes!

- ▶ Multi-Level Table of Contents
- ▶ Multi-Level Index
- ▶ Lists of Tables and Figures
- ▶ Numbers Paragraphs (1.2, 1.3, etc)
- ▶ Prepares Footnotes
- ▶ Prepares Bibliography
- ▶ Clear commands, Numerous formatting options
- ▶ Reads documents with nested files

Specify When Ordering; Operating System, Computer Type and Disk Format. Free brochure and nearly free demo disk available.

STARMATE

(Special Introductory Price)

Demo All Disk (credits to purchase) \$12.95

\$145.00

**SOLUTION
TECHNOLOGY, INC.**

"We Deliver Productivity"

1499 Palmetto Park Road
Suite 218
Boca Raton, FL 33432
305/368-6228

WordStar PC-DOS and CP/M are trademarks of
MicroPro, IBM and Digital Research respectively.

Check or COD, Florida residents add 5% sales tax.
Dealer and Distributor inquiries welcome.

Circle no. 61 on reader service card.

¹⁰ M. Zimmermann, "Big Numbers & Small Computers," *Creative Computing*, January 1982, pp. 126-138.

¹¹ J. Nyberg, "A Fast, Ancient Method of Multiplication," *Byte*, October 1981, pp. 376-377.

Additional References

1. B. Schanning, "Securing Data Inexpensively Via Public Keys," *Computer Design*, April 5, 1983, pp. 105-108.

2. J. Smith, "Public Key Cryptography," *Byte*, January 1983, pp. 198-218.

3. H. T. Gordon, "Fast Divisibility Algorithms," *DDJ*, June 1983, No. 80, pp. 14-16. **DDJ**

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 192.

RSA Cryptography System (Text begins on page 16)

Listing One

```
# *** Listing 1 -- an example of the RATFOR precompiler output
# ***

#           Note that the resulting FORTRAN has the spaces and new lines
#           deleted and that the result is much less readable than the
#           RATFOR source !!!

define(NEWVARIABLE,variable30) # look at Label 8 for the replacement

program example

    integer variable1,variable2,variable3,variable4,variable5,variable6,
        variable7,variable8,variable9,variable10

1   continue # separator for use in finding where statements begin in
        # generated FORTRAN code (standard FORTRAN statement)
repeat # a RATFOR Construct
    { # "{" and "}" are not needed for a single statement (see WHILE) !!!
        statement a
    }
until (variable1 == variable2)

2   continue
while (variable3 != variable4)
    statement b # in-line comment

3   continue
for (variable5=0.0; variable5 < variable6; variable5=variable5+1.0)
    {
        statement c
        if (variable7 >= variable8)
            {
                break
            }
    }

4   continue
do n=j,k # Note the lack of a terminating label number. RATFOR
        # will fill it in for you !!!
    {
        if (variable9 >= variable10)
            {
                next
            }
        statement d
    }

5   continue
if (variable7 > variable8)
```

(Continued on page 24)

**"The Sage IV
may well be
the best
68000-chip
computer on
the market"**

January 1984 © Byte Publications, Inc.

John-
Have you seen this?
let's discuss ASAP!
Bill

In January, Byte Magazine referred to the SAGE IV microcomputer as being "... a good candidate for the best available microcomputer, based on the 68000 chip".

We happen to agree and so do the thousands of SAGE users worldwide.

Let us tell you more about it.
Contact us at:

702-322-6868

or write us at 4905 Energy Way,
Reno, Nevada 89502.

SAGETM
COMPUTER

RSA Cryptography System (Listing continued, text begins on page 16)

Listing One

```
      { # "{" and "}" must be used here since ";" makes multiple lines !!!  
      statement e; statement f; statement g  
      }  
else if ((! logical1) & (variable9 <= variable10) |  
        (logical2 & logical3))  
      {  
      statement h  
      }  
else  
      {  
      statement i  
      }
```

include LISTING2 # insert Listing 2 here

End Listing One

Listing Two

```
# *** Listing 2 -- this file will be included into Listing 1 at the occurrence  
# ***           of the "include LISTING2" statement !!!
```

```
6      call subroutine(variable11,variable12,variable13,variable14,variable15,  
                      variable16,variable17,variable18,variable19,variable20)
```

```
7      variable21=variable22*variable23/variable24+variable25-  
          (variable26/variable27)+variable28
```

```
8      variable31=NEWVARIABLE
```

End Listing Two

Listing Three

```
C *** Listing 3 -- resultant FORTRAN source generated by passing
```

```
C ***           Listing1 & Listing2 through RATFOR
```

```
C ***
```

```
      programexample  
      integervariable1,variable2,variable3,variable4,variable5,variable6  
      *,variable7,variable8,variable9,variable10  
1      continue  
      continue  
23000 continue  
      statementa  
23001 if(.not.(variable1.eq.variable2))goto 23000  
23002 continue  
2      continue  
      continue  
23003 if(.not.(variable3.ne.variable4))goto 23004  
      statementb  
      goto 23003  
23004 continue  
3      continue  
      continue  
      variable5=0.0
```

(Continued on page 26)

Discover the wizardry of **MAGIC**

MAchine Generated Integrated Code®

MAGIC/MPS™—A revolution in program development. **MAGIC** supersedes COBOL, PASCAL, C, BASIC and other application languages. **MAGIC** enables you to produce good clean assembly language machine code—and plenty of it.

MAGIC through its various compilers allows the use of the same source to derive multiple object codes for a wide variety of systems.

MAGIC makes a run-time package a thing of the past. **MAGIC** lets you address diverse system and application needs in significantly less time.

For more information, send coupon or write Data Management Assoc., Inc.,
P.O. Box 4340, Wilmington DE 19807.
Phone 302/655-8986.

**MAGIC MAKES PROGRAMMING
EXCITING**



MAGIC is a language written for programmers
by programmers.

MPS/80™ for CP/M™ systems

MPS/86™ for CP/M-86™ and MS™-DOS systems

SHOW ME SOME MAGIC!

Mail coupon to Data Management Assoc., Inc., PO Box 4340,
Wilmington, Delaware 19807, or call 302/655-8986.

My system is _____

- Send: ☐ More information
☐ **MAGIC/MPS™** Manual @ \$50 ea.
☐ **MPS/80™** @ \$795 ea.
☐ **MPS/86™** @ \$995 ea.
☐ Check enclosed ☐ Send C.O.D.

Name _____

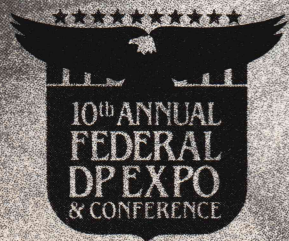
Title _____

Company _____

Address _____

City/State/Zip _____

Phone() _____ Best time to call _____



**LET US SHOW YOU
SOME MAGIC AT THE
FEDERAL DP EXPO**

RSA Cryptography System (Listing continued, text begins on page 16)

Listing Three

```
23005 if(.not.(variable5.lt.variable6))goto 23007
      statementc
      if(.not.(variable7.ge.variable8))goto 23008
      goto 23007
23008 continue
23006 variable5=variable5+1.0
      goto 23005
23007 continue
4      continue
      do 23010n=j,k
      if(.not.(variable9.ge.variable10))goto 23012
      goto 23010
23012 continue
      statementd
23010 continue
23011 continue
5      continue
      if(.not.(variable7.gt.variable8))goto 23014
      statemente
      statementf
      statementg
      goto 23015
23014 continue
      if(.not.((.not.logical1).and.(variable9.le.variable10).or.(logical
*2.and.logical3)))goto 23016
      statementh
      goto 23017
23016 continue
      statementi
23017 continue
23015 continue
6      callsubroutine(variable11,variable12,variable13,variable14,variabl
*e15,variabl16,variable17,variable18,variable19,variable20)
7      variable21=variable22*variable23/variable24+variable25-(variable26
*/variable27)+variable28
8      variable31=variable30
      end
```

End Listing Three

Listing Four

```
#####
###                                     ###
###      Copyright 1983, Charles E. Burton, Denver, Colorado      ###
###                                     ###
### All rights reserved.  Permission granted to use this software for ###
###   personal, non-commercial purposes only.                      ###
###                                     ###
#####

# PROGRAM NAME:  MPARITH.RAT
# PURPOSE:      Unsigned Multiple Precision Arithmetic Routines (re: D.E. Knuth,
#               The Art of Computer Programming, V. 2 (Semi-Numerical Algorithms),
#               2nd Ed., (Addison-Wesley, Reading, MA), pp. 250-268.)
#               MPADD  -- Multiple Precision Addition
#               MPSUBT -- Multiple Precision Subtraction
#               MPMULT -- Multiple Precision Multiplication
#               MPDIV  -- Multiple Precision Division
```



```

#
# LANGUAGE:  RATFOR
# AUTHOR:   CEB
# USAGE:   CALL MPADD (NUM1, LEN1, NUM2, LEN2, SUM, LENS, MODULO)
#          CALL MPSUBT (NUM1, LEN1, NUM2, LEN2, DIFF, LEND, MODULO)
#          CALL MPMULT (NUM1, LEN1, NUM2, LEN2, PROD, LENP, MODULO)
#          CALL MPDIV (NUM1, LEN1, NUM2, LEN2, QUOT, LENQ, REMN, LENR, MODULO)
#          <>NUM* -- Byte array, contains the number (byte modulus MODULO)
#                  to be operated on. The MSDigit(s) are in
#                  NUM*(1) and the LSDigit(s) are in NUM*(LEN*). NUM2 is
#                  never modified and NUM1 is only modified by MPDIV.
#                  CAUTIONS: for MPSUBT, NUM1 must be the larger number;
#                  for MPDIV, NUM1 must be the Numerator such that
#                  NUM1(1) == 0 (required for normalization) and NUM2
#                  must be the Demominator such that NUM2(1) != 0.
#          >LEN1/2 -- Integer variable, defines the length of the NUM*
#                  array.
#          <SUM -- Byte array, contains the sum (byte modulus MODULO)
#                  of NUM1 + NUM2. The MSDigit(s) are in
#                  SUM(1) and the LSDigit(s) are in SUM(LENS).
#          >LENS -- Integer variable, defines length of SUM array.
#                  CAUTION: LENS = MAX(LEN1, LEN2) + 1
#          <DIFF -- Byte array, contains the difference (byte modulus
#                  MODULO) of NUM1 - NUM2. The MSDigit(s) are in
#                  DIFF(1) and the LSDigit(s) are in DIFF(LEND).
#          >LEND -- Integer variable, defines length of DIFF array.

```

(Continued on next page)



VISIT OUR BOOTH AT THE WEST COAST
COMPUTER FAIRE

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. Detailed manual included. WRITE is \$239.00.

WORKMAN & ASSOCIATES

112 Marion Avenue
Pasadena, CA 91106
(213) 796-4401

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, NorthStar, Osborne, KayPro, Monroe, Otrona, Epson QX-10, NEC/PC, DEC VT-180, TI Professional, Access, Morrow. Please request our new catalog. We welcome COD orders.

Circle no. 69 on reader service card.

8748

CP/M SIMULATOR/DEBUGGER FOR THE INTEL 8748/8048

ANNOUNCING SIM48

- SIM48 allows you to load, trace, execute and save Intel 8748/8048 software using standard Intel Hex files.
- SIM48 allows you to simulate all instruction operations, timer/counter operations, I/O operations, interrupt processing, reset execution, internal and external RAM and ROM.
- SIM48's command set includes load, breakpoint, assemble, list (disassemble), trace, call and execute commands (as seen in DDT and ZSID).
- SIM48 allows you to simulate all software operations of the 8748/8048, yet costs 1/20th of an In-Circuit Emulator.
- SIM48 is CP/M compatible. Supplied on an 8" SSSD diskette.
- SIM22 (for Intel 8021/8022's) and SIM51 (for Intel 8751/8051's) soon to be released.

SIM48 \$150.00 SIM48 Manual \$20.00

Plus shipping and handling.
N.Y. State residents add sales tax.
Mastercard/Visa

Logical Systems

6184 TEALL STATION
SYRACUSE, NY 13217
(315) 457-9416

SIM48, SIM22, and SIM51 are trademarks of Logical Systems Corporation.
CP/M, ZSID, and DDT are trademarks of Digital Research.

Circle no. 29 on reader service card.

RSA Cryptography System (Listing continued, text begins on page 16)

Listing Four

```
#          CAUTION:  LEND = LEN1
#
# <PROD -- Byte array, contains the product (byte modulus MODULO)
#          of NUM1 * NUM2.  The MSDigit(s) are in
#          PROD(1) and the LSDigit(s) are in PROD(LENP).
# >LENP -- Integer variable, defines length of PROD array.
#          CAUTION:  LENP = LEN1 + LEN2
#
# <QUOT -- Byte array, contains the quotient (byte modulus MODULO)
#          of NUM1 / NUM2.  The MSDigit(s) are in
#          QUOT(1) and the LSDigit(s) are in QUOT(LENQ).
# >LENQ -- Integer variable, defines length of QUOT array.
#          CAUTION:  LENQ = MAX(LEN1 - LEN2, 1)
#
# <REMN -- Byte array, contains the remainder (byte modulus
#          MODULO) of NUM1 / NUM2.  The MSDigit(s) are in
#          REMN(1) and the LSDigit(s) are in REMN(LENR).
# >LENR -- Integer variable, defines length of REMN array.
#          CAUTION:  LENR = LEN2
#
# >MODULO -- Integer variable, defines the arithmetic modulus that
#            is to be used.  MODULO has a byte-wide effect and
#            should be between 2 and 128 (e.g. 100 for Decimal
#            Numbers and 128 for ASCII Characters).
#            CAUTION:  The Arrays MUST have the same modulus and
#            MUST be positive (unsigned) for proper operation !!!
#
#
# ARRAYS USED:  NUM1(*), NUM2(*), SUM(*), DIFF(*), PROD(*), QUOT(*), REMN(*)
# EXTERNALS:
# UPDATE HISTORY:  INITIAL RELEASE -- 01/18/83 CEB
#
```

```
subroutine mpadd(num1,len1,num2,len2,sum,lens,modulo)
```

```
byte num1(1),num2(1),sum(1)
integer len1,len2,lens,modulo,add,carry

if (lens >= max0(len1,len2)+1) # SUM array have sufficient length?
(
  idxsum=lens # get the index to the LSDigit of SUM
  idxn1=len1 # get index to LSDigit of NUM1
  idxn2=len2 # get index to LSDigit of NUM2
  carry=0 # initialize CARRY
  while (min0(idxn1,idxn2) > 0) # look at each packed character
  (
    itemp1=num1(idxn1); itemp2=num2(idxn2) # byte to integer
    add=carry+itemp1+itemp2 # get sum
    sum(idxsum)=mod(add,modulo) # generate SUM value
    carry=add/modulo # generate CARRY for next pass
    idxn1=idxn1-1 # move indices to next position
    idxn2=idxn2-1
    idxsum=idxsum-1
  )
  if (idxn2 == 0) # 1st number still has data?
  (
    while (idxn1 > 0) # look at rest of 1st number
    (
      itemp1=num1(idxn1) # byte to integer
      add=carry+itemp1 # get sum
      sum(idxsum)=mod(add,modulo) # generate SUM value
```

(Continued on page 30)

Don't call her cheap. Call her beautiful.

The Bonnie BlueTM

Word Processing System for the IBM Personal Computer

It's obvious what makes her so cheap, but what makes Bonnie Blue so beautiful? Bonnie Blue is a new and easy-to-use word processing program for the IBM Personal Computer.

The Full System. The Bonnie Blue System includes in one program a full screen Editor, a Printing module and a useful Toolbox. It includes the features you've come to expect, and more:

complete cursor control: by character, word, line; page up and down instantly; go to top, bottom of document; auto scroll towards top or bottom

word wrap

margin justification, centering

adjustable margins, tabs, indents

reformat paragraphs

move, copy, delete, paste blocks

find with delete, insert, replace and wild card characters

keyboard remapping

multi-line headers, footers

Bonnie Blue can handle lines longer than the screen is wide, by horizontally scrolling the line. And, unlike some programs, Bonnie Blue lets you include any displayable character in your text, such as block graphics and foreign language characters.

Unique Features. With Bonnie Blue, you can "paint" display attributes onto your text, by the character, word, or line, or automatically as you enter text. With the monochrome adapter, you can paint any combination of underlined, bold, reverse video or blinking. With an 80 column monitor and the color/graphics adapter, this translates into a palette of 16 color combinations to choose from. And if your computer has both monitors, Bonnie Blue lets you use them both, shifting back and forth as you wish.

Powerful Printing Module. You can use these colors or display attributes to highlight text on the screen, and Bonnie Blue can remove them from a file when you want (all files created by Bonnie Blue are DOS standard). The Printing module understands these text attributes, and you can map them into any single printer function or combination.

For example, normally you would want underlined text to print underlined. But you can tell Bonnie Blue to print underlined characters as both underlined and bold. Bright text on the screen can mean double struck, or emphasized and in italics. You are at the controls.

The first Print formatting module supports all the text capabilities of the Epson MX series with Grafrax Plus. By the time this ad appears, we will be supporting other popular dot-matrix and letter quality printers.

More than thirty "dot" commands give you added control of the format of your finished document. You can send it to a disk file instead of the printer, or preview the final page formatting on the screen.

Toolbox. The Toolbox is a set of useful functions, called "filters" that allow you to extract information from your files and transform their content. With these tools, you can join files together, sort lines of text, count words, find and substitute patterns, etc. Writers and programmers find this a useful collection of productivity enhancers.

Bonnie Blue is also great for a hard disk system. A thorough User's Guide, complemented by help screens and roadmaps, make the Bonnie Blue an exceptionally easy-to-learn and easy-to-use system.

Order yours today, or send for our free brochure. Bonnie Blue is available exclusively from **Bonnie Blue Software**, P.O. Box 536, Liverpool, NY 13088.

IBM Personal Computer is a trademark of IBM Corp. Epson Grafrax Plus is a trademark of Epson America Inc.

Bonnie Blue Software

Post Office Box 536
Liverpool, NY 13088

☐ Send me the Bonnie Blue System. I am enclosing \$50 (NY State residents please add 7% sales tax).

☐ Please send literature. I have a _____

☐ Check enclosed ☐ VISA ☐ MasterCard Sorry, no COD.

Credit Card No. _____ Expires _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Company _____

Only \$50

Minimum

recommended system:

IBM PC, 128K, 2 disk drives,
PC-DOS 1.1 or 2.0, 80-column
monitor or monochrome adapter,
or both, Epson MX-80 or
MX-100 with Grafrax Plus.

*Versions available soon for PCjr.
Write for details.*

RSA Cryptography System (Listing continued, text begins on page 16)

Listing Four

```
        carry=add/modulo # generate CARRY for next pass
        idxn1=idxn1-1 # move indices to next position
        idxsum=idxsum-1
    }
}
else # 2nd number still has data?
{
    while (idxn2 > 0) # look at rest of 2nd number
    {
        itemp1=num2(idxn2) # byte to integer
        add=carry+itemp1 # get sum
        sum(idxsum)=mod(add,modulo) # generate SUM value
        carry=add/modulo # generate CARRY for next pass
        idxn2=idxn2-1 # move indices to next position
        idxsum=idxsum-1
    }
}
sum(idxsum)=carry # put final CARRY in SUM
idxsum=idxsum-1 # move index next position
while (idxsum > 0) # finish filling in SUM
{
    sum(idxsum)=0 # zero out remainder of SUM
    idxsum=idxsum-1 # move index to next position
}
}
else # cannot generate SUM
{
    call rmrkln('?.')
    call error('Length of [SUM] too small !!!.') # print message & exit
}
return

end

#####

subroutine mpsubt(num1,len1,num2,len2,diff,lend,modulo)

    byte num1(1),num2(1),diff(1)
    integer len1,len2,lend,modulo,subt,borrow

    if ((lend >= len1) & (len1 >= len2)) # DIFF array have sufficient length
                                         # and NUM1 at least as big as NUM2?
    {
        idxdif=lend # get the index to the LSDigit of DIFF
        idxn1=len1 # get index to LSDigit of NUM1
        idxn2=len2 # get index to LSDigit of NUM2
        borrow=0 # initialize BORROW
        while (idxn2 > 0) # look at each packed character
        {
            itemp1=num1(idxn1); itemp2=num2(idxn2) # byte to integer
            subt=borrow+itemp1-itemp2 # get difference
            if (subt < 0) # need to do MODULO's complement?
            {
                subt=modulo+subt # do MODULO's complement
                borrow=-1 # indicate borrow
            }
            else # everything is okay
        }
    }
}
```



```

        borrow=0
        diff(idxdif)=subt # generate DIFF value
        idxn1=idxn1-1 # move indices to next position
        idxn2=idxn2-1
        idxdif=idxdif-1
    }
while (idxn1 > 0) # look at rest of 1st number
{
    itemp1=num1(idxn1) # byte to integer
    subt=borrow+itemp1 # get difference
    if (subt < 0) # need to do MODULO's complement?
    {
        subt=modulo+subt # do MODULO's complement
        borrow=-1 # indicate borrow
    }
    else # everything is okay
        borrow=0
    diff(idxdif)=subt # generate DIFF value
    idxn1=idxn1-1 # move indices to next position
    idxdif=idxdif-1
}
if (borrow == 0) # NUM1 >= NUM2?
{
    while (idxdif > 0) # finish filling in DIFF
    {
        diff(idxdif)=0 # zero out remainder of DIFF
    }
}

```

(Continued on next page)

C COMPILER

- FULL C
- UNIX* Ver. 7 COMPATABILITY
- NO ROYALTIES ON GENERATED CODE
- GENERATED CODE IS REENTRANT
- C AND ASSEMBLY SOURCE MAY BE INTERMIXED
- UPGRADES & SUPPORT FOR 1 YEAR

C SOURCE AVAILABLE FOR \$2500⁰⁰

HOST	6809 TARGET	PDP-11*/LSI-11* TARGET	8080/(Z80) TARGET	8088/8086 TARGET
FLEX*/UNIFLEX* OS-9*	\$200.00 \$350.00	500.00	500.00	500.00
RT-11*/RSX-11* PDP-11*	500.00	200.00 350.00	500.00	500.00
CP/M* 8080/(Z80)	500.00	500.00	200.00 350.00	500.00
PCDOS*/CP/M86* 8088/8086	500.00	500.00	500.00	200.00 350.00

*PCDOS is a trademark of IBM Corp. MSDOS is a trademark of MICROSOFT. UNIX is a trademark of BELL LABS. RT-11/RSX-11/PDP-11 is a trademark of digital Equipment Corporation. FLEX/UNIFLEX is a trademark of Technical Systems consultants. CP/M and CP/M86 are trademarks of Digital Research. OS-9 is a trademark of Microware & Motorola.

408-275-1659

TELECON SYSTEMS
1155 Meridian Avenue, Suite 218
San Jose, California 95125

Circle no. 64 on reader service card.

WOW!

P-R-O-P-O-R-T-I-O-N-A-L Spacing on WordStar

You are reading text printed by WordStar in proportional spacing, providing a professional, **easy to read**, typeset appearance.

Complete details for printing in proportional spacing directly from WordStar, setting two or more fully justified columns on a page, and underlining spaces

between words, are provided. The techniques will work on all versions of WordStar, and will drive Diablo, Xerox, Qume, NEC, and other daisy-wheel printers.

Above text printed on a daisywheel printer direct from WordStar.

Now you can have the professional appearance of typeset text. Using PS is as easy as turning on bold or underline and is done right in your document, then printed by WordStar automatically!

PS ON WordStar-\$20 postpaid

Please send me _____ Copies. Enclosed is my check (or Visa/MC# and exp. date) for \$_____, made out to:

WRITING CONSULTANTS

Suite 165 / 11 Creek Bend Drive
Fairport, New York 14450

Orders Only, Call Toll Free 1-800-227-3800 Ext. 7018
Dealer inquiries invited.

Circle no. 70 on reader service card.

RSA Cryptography System

(Listing continued, text begins on page 16)

Listing Four

```
        idxdif=idxdif-1 # move index to next position
    }
}
else # NUM1 < NUM2 (error)
{
    call rmrkln('.')
    call error('[NUM1] < [NUM2] in subtraction.') # print message
                                                # & exit
}
}
else # cannot generate DIFF
{
    call rmrkln('.')
    call error('Length of [DIFF] too small or [NUM1] < [NUM2] !!!.')
                                                # print message & exit
}
return
end

#####

subroutine mpmult(num1,len1,num2,len2,prod,lenp,module)

    byte num1(1),num2(1),prod(1)
    integer len1,len2,lenp,module,mult,carry

    if (lenp >= len1+len2) # PROD array have sufficient length?
    {
        idxn2=len2 # get index to LSDigit of NUM2
        do idxprd=1,lenp # clean out PROD
            prod(idxprd)=0
        while (idxn2 > 0) # multiply numbers
        {
            idxprd=lenp+(idxn2-len2) # get index to PROD
            if (num2(idxn2) == 0) # multiplier zero?
            {
                carry=0 # set CARRY (product) to zero
                idxprd=idxprd-len1 # get index to PROD
            }
            else # finite multiplier value
            {
                idxn1=len1 # get index to LSDigit of NUM1
                carry=0 # initialize CARRY
                while (idxn1 > 0) # multiply a "digit" at a time
                {
                    itemp1=num1(idxn1); itemp2=num2(idxn2) # byte to
                    itemp3=prod(idxprd) # integer
                    mult=carry+itemp1*itemp2+itemp3 # get product
                    prod(idxprd)=mod(mult,module) # generate new product
                                                # value
                    carry=mult/module # generate CARRY for next pass
                    idxn1=idxn1-1 # move indices to next position
                    idxprd=idxprd-1
                }
            }
            prod(idxprd)=carry # save CARRY for next pass
        }
    }
}
```

(Continued on page 34)

programmers

READ THIS...

NOW, I KNOW I CAN MAKE BIG MONEY WRITING AND SELLING MY PROGRAMS. THIS BOOK TOLD ME WHAT TO WRITE — WHO TO SELL IT TO — THOUSANDS OF NAMES, ADDRESSES, IDEAS, GUIDELINES. "SOFTWARE WRITER'S MARKET" IS A FANTASTIC BOOK!



- * WHO TO SELL YOUR PROGRAMS TO
- * THOUSANDS OF COMPANY NAMES AND ADDRESSES, WITH DETAILED LISTINGS SHOWING:
 - (1) WHAT PROGRAMS PUBLISHERS ARE LOOKING FOR
 - (2) HOW THEY WANT YOU TO SUBMIT YOUR PROGRAM
 - (3) HOW MUCH THEY PAY — AND WHEN!
- * 100 CATEGORIES — FROM "ACCOUNTS RECEIVABLE" TO "GAMES" TO "VIDEO CONTROL" PROGRAMS
- * HOW TO WRITE CLEAR DOCUMENTATION
- * DEBUGGING TECHNIQUES

Enclose check or money order
for \$19.95 (No C.O.D.'s) to:

IPF Publications
146 Country Club Lane
Pomona, NY 10970
(914) 354-5585

Order Your Copy Today!

Name

Address

City..... State..... Zip.....

RSA Cryptography System (Listing continued, text begins on page 16)

Listing Four

```
        idxn2=idxn2-1 # move index to NUM2 to next position
    }
}
else # cannot generate PROD
{
    call rmrkl('..')
    call error('Length of [PROD] too small !!!.') # print message & exit
}
return
end

#####

subroutine mpdiv(num1,len1,num2,len2,quot,lenq,remn,lenr,modulo)

    byte num1(1),num2(1),quot(1),remn(1)
    integer len1,len2,lenq,lenr,modulo,div,mult,add,borrow,carry,flag,scale,
        qtest

    equivalence (borrow,carry,flag),(div,mult,add)

    if ((lenq >= len1-len2) & (lenr >= len2) & (num1(1) == 0) &
        (num2(1) != 0)) # QUOT & REMN arrays have sufficient length and
        # valid numerator & denominator?
    {
        do idxquo=1,lenq # clean out QUOT
            quot(idxquo)=0
        do idxrem=1,lenr # clean out REMN
            remn(idxrem)=0
        flag=0 # initialize numerator zero flag
        do idxn1=2,len1 # check numerator for Zero
            {
                if (num1(idxn1) != 0) # non-zero "digit" found?
                {
                    flag=1 # indicate non-zero numerator
                    break # exit DO loop
                }
            }
        }

        #
        # could check for Numerator <= Denominator here !!!
        # if (Numerator == Denominator)
        #     QUOT = 1
        #     REMN = 0
        # else if (Numerator < Denominator)
        #     QUOT = 0
        #     REMN = Numerator
        # else
        #     continue
        #
        if (flag != 0) # non-zero numerator?
        {
            itemp1=num2(1) # byte to integer
            scale=modulo/(itemp1+1) # get normalizing scale factor
            if (scale > 1) # normalization required?
            {
                carry=0 # initialize CARRY
                idxn1=len1 # initialize numerator pointer
                while (idxn1 > 0) # normalize numerator
```

(Continued on page 36)

COHERENT™ IS SUPERIOR TO UNIX* AND IT'S AVAILABLE TODAY ON THE IBM PC.

Mark Williams Company hasn't just taken a mini-computer operating system, like UNIX, and ported it to the PC. We wrote COHERENT ourselves. We were able to bring UNIX capability to the PC with the PC in mind, making it the most efficient personal computer work station available at an unbelievable price.

For the first time you get a multi-user, multitasking operating system on your IBM PC. Because COHERENT is UNIX-compatible, UNIX software will run on the PC under COHERENT.

The software system includes a C-compiler and over 100 utilities, all for \$500. Similar environments cost thousands more.

COHERENT on the IBM PC requires a hard disk and 256K memory. It's available on the IBM XT, and Tecmar, Davong and Corvus hard disks.

Available now. For additional information, call or write,

Mark Williams Company
1430 West Wrightwood, Chicago, Illinois 60614
312/472-6659



COHERENT is a trade mark of Mark Williams Company.

*UNIX is a trade mark of Bell Laboratories.

Listing Four

```
{
  itemp1=num1(idxn1) # byte to integer
  mult=carry+scale*itemp1 # get product
  num1(idxn1)=mod(mult,module) # generate new product
                                # value
  carry=mult/module # generate carry for next pass
  idxn1=idxn1-1 # move index to next position
}
carry=0 # initialize CARRY

idxn2=len2 # initialize denominator pointer
while (idxn2 > 0) # normalize denominator
{
  itemp1=num2(idxn2) # byte to integer
  mult=carry+scale*itemp1 # get product
  num2(idxn2)=mod(mult,module) # generate new product
                                # value
  carry=mult/module # generate carry for next pass
  idxn2=idxn2-1 # move index to next position
}
}
# else NUM1(1) == 0, so scaling is already done
idxn1=1 # initialize index of NUM1
while (idxn1 <= (len1-len2)) # calculate quotients
{
  if (num2(1) == num1(idxn1)) # get initial test quotient
    qtest=modulo-1
  else
  {
    itemp1=num1(idxn1); itemp3=num2(1) # byte to integer
    if ((idxn1+1) <= len1) # valid index?
      itemp2=num1(idxn1+1)
    else # no
      itemp2=0
    qtest=(modulo*itemp1+itemp2)/itemp3
  }
  itemp2=num1(idxn1); itemp4=num2(1) # byte to integer
  if (len2 >= 2) # valid index?
    itemp1=num2(2)
  else # no
    itemp1=0
  if ((idxn1+1) <= len1) # valid index?
  {
    itemp3=num1(idxn1+1)
    if ((idxn1+2) <= len1) # valid index?
      itemp5=num1(idxn1+2)
    else # no
      itemp5=0
  }
  else # no
  {
    itemp3=0
    itemp5=0
  }
  while (qtest*itemp1 > (modulo*(modulo*itemp2+itemp3-
    qtest*itemp4)+itemp5))
    # check if test quotient is too large
```



```

    {
        qtest=qtest-1
    }
    idxn2=len2 # initialize index to NUM2
    idx=idxn1+len2 # initialize index to NUM1
    borrow=0 # initialize BORROW

while (idx >= idxn1) # perform multiply & subtract
{
    if (idxn2 > 0) # another "digit" of NUM2?
    {
        itemp1=num2(idxn2) # byte to integer
        borrow=borrow-qtest*itemp1
    }
    itemp1=num1(idx) # byte to integer
    div=borrow+itemp1 # generate mult. & subt.
    if (div < 0) # need to do MODULO's complement?
    {
        borrow=div/modulo # get BORROW for next pass
        div=mod(div,modulo) # MODULO's complement
        if (div < 0) # still need to make DIV positive?
        {
            div=modulo+div # make it positive
            borrow=borrow-1 # adjust borrow
        }
    }
    else # everything is okay
        borrow=0
    num1(idx)=div # update numerator
    idxn2=idxn2-1 # move indices to next position

    idx=idx-1
}
idxquo=lenq-(len1-len2)+idxn1 # get index to QUOT
if (borrow != 0) # need to add back divisor?
{
    quot(idxquo)=qtest-1 # adjust quotient
    idxn2=len2 # get index to NUM2
    idx=idxn1+len2 # get index to NUM1
    carry=0 # initialize CARRY
    while (idx >= idxn1) # add back
    {
        if (idxn2 > 0) # another "digit" of NUM2?
        {
            itemp1=num2(idxn2) # byte to integer
            carry=carry+itemp1 # add it back
        }
        itemp1=num1(idx) # byte to integer
        add=carry+itemp1 # add it back in
        if (add >= modulo) # need to adjust ADD?
        {
            add=add-modulo # pull ADD back into range
            carry=1 # set CARRY for next pass
        }
        else # no adjustment
            carry=0 # no carry
        num1(idx)=add # update numerator
    }
}

```

(Continued on next page)

RSA Cryptography System (Listing continued, text begins on page 16)

Listing Four

```
        idxn2=idxn2-1 # move indices to next position
        idx=idx-1
    }
}

    else # no add back needed
        quot(idxquo)=qtest # generate quotient
        idxn1=idxn1+1 # move index to next position
    }
    idxn1=len1-len2+1 # get index to NUM1
    idxrem=lenr-len2+1 # get index to REMN
    borrow=0 # initialize BORROW
    while (idxn1 <= len1) # unnormalize remainder
    {
        itemp1=num1(idxn1) # byte to integer
        div=itemp1+modulo*borrow # get numerator of REMN
        remn(idxrem)=div/scale # remove normalization
        borrow=mod(div,scale) # get borrow for next pass
        idxn1=idxn1+1 # move indices to next position
        idxrem=idxrem+1
    }
    idxn2=1 # get index to NUM2
    borrow=0 # initialize BORROW
    while (idxn2 <= len2) # unnormalize NUM2
    {
        itemp1=num2(idxn2) # byte to integer
        div=itemp1+modulo*borrow # get numerator of NUM2
        num2(idxn2)=div/scale # remove normalization
        borrow=mod(div,scale) # get borrow for next pass
        idxn2=idxn2+1 # move index to next position
    }
}

else # cannot generate QUOT & REMN
{
    call rmrkln('.')
    call remark('Length of [QUOT] or [REMN] too small or .')
    call error('invalid numerator or denominator !!!')
    # print message and exit
}
return

end
```

End Listing Four

Listing Five

```
#####
###                                     ###
###      Copyright 1983, Charles E. Burton, Denver, Colorado      ###
###                                     ###
### All rights reserved.  Permission granted to use this software for ###
###   personal, non-commercial purposes only.                     ###
###                                     ###
#####

# PROGRAM NAME:  RPEASANT.RAT
# PURPOSE:  Russian Peasant exponentiation (re. D.E. Knuth,
```



```
#
#
#
#
# LANGUAGE:  RATFOR
# AUTHOR:   CEB
# USAGE:    CALL RPEXP(NUM1,LEN1,NUM2,LEN2,EXPONENT,LENE,MODL,LENM,WORK,LENW,MODULO)
#
#           <>NUM* -- Byte array, contains the number [NUM* mod MODL] (byte
#                   modulus MODULO) to be operated on. The MSDigit(s) are
#                   in NUM*(1) and the LSDigit(s) are in NUM*(LEN*).
#                   NUM2 is never modified, but NUM1 is always modified !
#           >LEN1/2 -- Integer variable, defines the length of the NUM*
#                   array.
#                   CAUTION:  LEN1 = LENM
#           <>EXPONENT -- Byte array, contains the exponentiation (byte modulus
#                   MODULO) of [NUM1 ** NUM2 mod MODL]. The MSDigit(s) are
#                   in EXPONENT(1) and the LSDigit(s) are in EXPONENT(LENE).
#           >LENE -- Integer variable, defines length of EXPONENT array.
#                   CAUTION:  LENE = MAX(LEN1, LENM)
#           >MODL -- Byte array, contains the modulus of the arithmetic to
#                   be used in the calculations (byte modulus MODULO).
#                   The MSDigit(s) are in MODL(1) and the LSDigit(s) are
#                   in MODL(LENM).
#                   CAUTION:  MODL(1) != 0
#           >LENM -- Integer variable, defines length of MODL array.
#           <>WORK -- Byte array, a working array needed to do modulus
#                   arithmetic using MODL, i.e. [ ? mod MODL].
```

(Continued on next page)

Six Times Faster!

Super Fast Z80 Assembly Language Development Package

Z80ASM

- Complete Zilog Mnemonic set
- Full Macro facility
- Plain English error messages
- One or two pass operation
- Over 6000 lines/minute
- Supports nested INCLUDE files
- Allows external bytes, words, and expressions (EXT1 * EXT2)
- Labels significant to 16 characters even on externals (SLR Format Only)
- Integral cross-reference
- Upper/lower case optionally significant
- Conditional assembly
- Assemble code for execution at another address (PHASE & DEPHASE)
- Generates COM, HEX, or REL files
- COM files may start at other than 100H
- REL files may be in Microsoft format or SLR format
- Separate PROG, DATA & COMMON address spaces
- Accepts symbol definitions from the console
- Flexible listing facility includes TIME and DATE in listing (CP/M Plus Only)

SLRINK

- Links any combination of SLR format and Microsoft format REL files
- One or two pass operation allows output files up to 64K
- Generates HEX or COM files
- User may specify PROG, DATA, and COMMON loading addresses
- COM may start at other than 100H
- HEX files do not fill empty address space.
- Generate inter-module cross-reference and load map
- Save symbol table to disk in REL format for use in overlay generation
- Declare entry points from console
- The FASTEST Microsoft Compatible Linker available

SPEED!
STRIPPED!
SPEED!

- Complete Package Includes: Z80ASM, SLRINK, SLRIB - Librarian and Manual for just \$199.99. Manual only, \$30.
- Most formats available for Z80 CP/M, CDOS, & TURBODOS
- Terms: add \$3 shipping US, others \$7. PA add 6% sales tax

For more information or to order, call:

1-800-833-3061

In PA, (412) 282-0864

Or write: SLR SYSTEMS

1622 North Main Street, Butler, Pennsylvania 16001

SLR Systems

RSA Cryptography System (Listing continued, text begins on page 16)

Listing Five

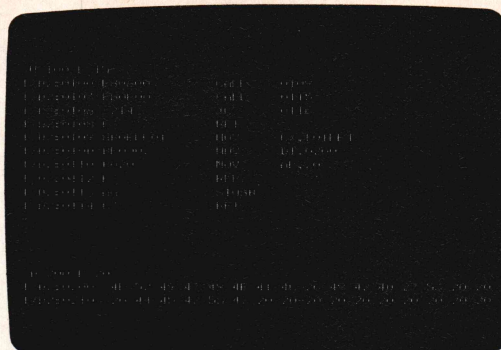
```
# >LENW -- Integer variable, defines length of WORK array.
# CAUTION: LENW = (LEN2 + 1) <NUM2 temp.>
# + (LEN1 + LENE) + 1 <Numerator>
# + MAX(LENE - LENM, 1) <Quotient>
# + LENM <Remainder>
# >MODULO -- Integer variable, defines the arithmetic byte modulus
# that is to be used. MODULO has a byte-wide effect
# and should be between 2 and 128 (e.g. 100 for Decimal
# Numbers and 128 for ASCII Characters).
# CAUTION: The Arrays MUST have the same modulus and
# MUST be positive (unsigned) for proper operation !!!
#
# ARRAYS USED: NUM1(*), NUM2(*), EXPN(*), MODL(*), WORK(*)
# EXTERNALS: MPMULT, MPDIV
# UPDATE HISTORY: INITIAL RELEASE -- 01/20/83 CEB
#
subroutine rpexp(num1,len1,num2,len2,expn,lene,modl,lenm,work,lenw,modulo)

byte num1(1),num2(1),expn(1),modl(1),work(1)
logical maskit,bit,temp,nextf
integer len1,len2,lene,lenm,lenw,modulo,mask,basmod

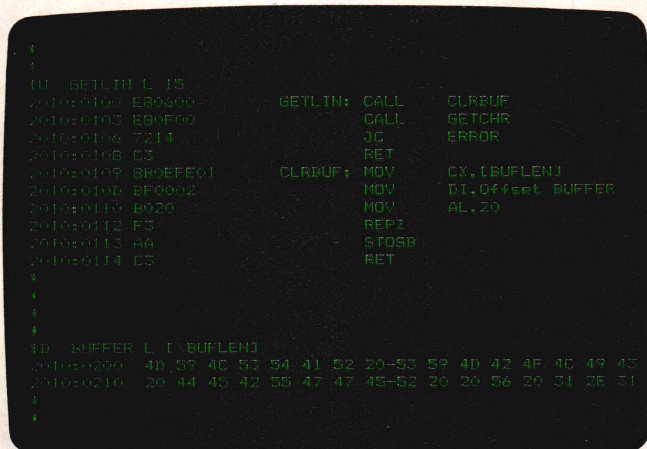
if ((len1 >= lenm) & (lene >= max0(len1,lenm)) &
    (lenw >= (len2+1)+(len1+lene+1)+max0(lene-lenm,1)+lenm))
    # NUM1, EXPN & WORK arrays have sufficient lengths?
    {
        basmod=1 # initialize basic modulus
        repeat # find basic modulus
            basmod=2*basmod # advance basic modulus
        until ((basmod == 128) ! (modulo/basmod == 1)) # basic modulus found
        do idxexp=1,lene # initialize EXPN array
            expn(idxexp)=0
        expn(lene)=1 # set EXPN=1
        do idxn2=1,len2 # copy NUM2 to temporary NUM2
            work(idxn2+1)=num2(idxn2)
        work(1)=0 # make sure it starts with zero (for MAKBIN routine)
        idxn2=len2 # get index to NUM2
        nextf=.true. # initialize next pass flag
        while (nextf) # generate Russian Peasant exponential
            {
                call makbin(work,idxwrk,temp,basmod,modulo,len2,idxn2,nextf)
                # make temporary NUM2 binary
            }
        mask=1 # initialize bit mask
        while (mask < basmod) # run through bits within each byte
            # of NUM2
            {
                maskit=mod(mask,256) # get LSByte of MASK
                bit=temp & maskit # get appropriate bit of NUM2
                if (bit != 0) # need to multiply NUM1 by EXPN?
                    call prdmod(expn,lene,num1,len1,modl,lenm,
                        work(idxwrk),modulo)
                # generate EXPN = (EXPN*NUM1) mod MODL
                call prdmod(num1,len1,num1,len1,modl,lenm,work(idxwrk),
                    modulo) # generate NUM1 = (NUM1*NUM1) mod MODL
                mask=2*mask # move to next mask bit
            }
    }
}
```

(Continued on page 42)

The difference



Original IBM Debug Program



Mylstar Symbolic Debugger V1.1

is Mylstar's Symbolic Debugging Program*

The plain and simple difference is that Mylstar's Symbolic Debugging Program speaks to your IBM PC in a language you both can understand, plain and simple.

Employing the same command structure, it allows you to use symbol names, mathematical expressions, batch files, on-line help, multi-command macros and other time-saving entries.

It's the enhancement to the *IBM Debug Program* you've been looking for—because it fills in the gaps—shortening the frustrating debugging process by as much as 50%—leaving you more time to do the work you need to do and the work you want to do, plain and simple.

Mylstar's Symbolic Debugging Program has been programmer-tested for over a year at Mylstar Electronics, Inc., (formerly D. Gottlieb & Co.), designers of the video arcade game, Q*BERT™.

TO ORDER...
Call (312) 562-7400 or mail coupon today.

*Designed for IBM PC-DOS 1.1 with 128K RAM minimum



**MYLSTAR
ELECTRONICS
INC.**

**165 West Lake Street
Northlake, Illinois 60164**

A Columbia Pictures Industries Company

**Mylstar Electronics, Inc.,
165 W. Lake St., Northlake, IL 60164**

Please send me Mylstar's Symbolic Debugging Program for use with the IBM PC computer. Enclosed is \$125, plain and simple.

☐ Check ☐ Money Order

118

NAME _____

FIRM _____

ADDRESS _____

CITY _____

STATE _____

ZIP _____

Illinois residents add 7% sales tax
Allow 2-4 weeks for delivery

Circle no. 37 on reader service card.

RSA Cryptography System (Listing Continued, text begins on page 16)

Listing Five

```
        idxn2=idxn2-1 # move to next NUM2 Digit
    }
}
else # cannot generate Russian Peasant Product
{
    call rmrkln('.')
    call error('Length of [NUM1], [EXPN] or [WORK] too small !!!.')
    # print message & exit
}
return

end

#####

# PRDMOD -- Product Modulo Routine:  NUM1 = (NUM1*NUM2) mod MODL

subroutine prdmod(num1,len1,num2,len2,mod1,lenm,work,module)

    byte num1(1),num2(1),mod1(1),work(1)
    integer len1,len2,lenm,module

    lenp=len1+len2 # initialize length of product of NUM1 & NUM2
    call mpmult(num1,len1,num2,len2,work,lenp,module) # multiply NUM1 & NUM2
    if (work(1) /= 0) # need to shift result for normalization in Division?
    {
        idxsum=lenp # initialize index to product
        while (idxsum > 0) # shift result down 1 byte
        {
            work(idxsum+1)=work(idxsum)
            idxsum=idxsum-1 # move index to next position
        }
        work(1)=0 # set MSDigit to zero
        lenp=lenp+1 # advance sum length
    }

    idxquo=lenp+1 # initialize pointer to start of Quotient array
    lenq=max0(lenp-lenm,1) # get Quotient length
    idxrem=idxquo+lenq # initialize pointer to start of Remainder array
    call mpdiv(work,lenp,mod1,lenm,work(idxquo),lenq,work(idxrem),lenm,module)
        # generate (NUM1*NUM2) mod MODL
    idxend=idxrem+lenm-1 # initialize index to end of remainder
    idxn1=len1 # get index to NUM1
    while (idxend >= idxrem) # remainder digits are available
    {
        num1(idxn1)=work(idxend) # move remainder to NUM1
        idxn1=idxn1-1 # move indices to next position
        idxend=idxend-1
    }
    while (idxn1 > 0) # any Digits left in NUM1
    {
        num1(idxn1)=0 # zero it
        idxn1=idxn1-1 # move index to next position
    }
    return

end

#####

# MAKBIN -- make sure Temporary NUM2 is binary
```



```

subroutine makbin(work,idxwrk,temp,basmod,modulo,len2,idxn2,nextf)

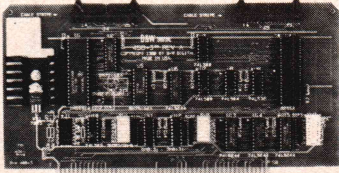
byte work(1),bmod
logical temp,nextf
integer idxwrk,basmod,modulo,len2,idxn2

lenw=len2+1 # get length of work
nextf=.false. # initialize next pass flag
if (basmod /= modulo) # NUM2 modulus not binary?
{
  idxquo=lenw+1 # get index to quotient
  lenq=len2 # get length of quotient
  idxrem=idxquo+lenq # get index to remainder
  bmod=basmod # integer to byte
  call mpdiv(work,lenw,bmod,1,work(idxquo),lenq,work(idxrem),1,modulo)
                                     # make temp. NUM2 binary
  temp=work(idxrem) # get binary temporary NUM2
  idxquo=idxquo+lenq-1 # get index to quotient
  idxwrk=lenw # get index to work
  while (idxquo > lenw) # move quotient to temporary NUM2 for next pass
  {
    if (work(idxquo) /= 0) # still digits left?
      nextf=.true. # flag a next pass
    work(idxwrk)=work(idxquo)
    idxquo=idxquo-1 # move indices to next position
    idxwrk=idxwrk-1
  }
  while (idxwrk > 0) # zero out remainder of temporary NUM2
  {
    work(idxwrk)=0
    idxwrk=idxwrk-1 # move index to next position
  }
}
else # NUM2 modulus is binary
{
  if (idxn2 > 1) # still digits left?
    nextf=.true. # flag a next pass
  temp=work(idxn2+1) # get binary temporary NUM2
}
idxwrk=lenw+1 # get index to start of unused work area
return

end

```

End Listing



THE 488+3 IEEE 488 TO S-100 INTERFACE

IEEE-488

- Handles all IEEE-488 1975/78 functions
- IEEE 696 (S-100) compatible
- MBASIC subroutines supplied; no BIOS mods required
- 3 parallel ports (8255A-5)
- Industrial quality; burned in and tested
- \$375

(Dealer inquiries invited)

D&W DIGITAL

20655 Hathaway Ave.
Hayward, CA 94541 415/ 887-5711

S-100

Circle no. 20 on reader service card.

Introduction to PL/C:

Programming Language for Compilers

As LSI densities and chip yields increased steadily, it became apparent that our ability to produce new and more powerful processors exceeded our ability to produce supporting software. An examination of technology advances indicates that hardware gains are due as much to the development of an appropriate tool technology as to the amount of direct resources expended. Efforts in the software domain, however, appear to have been directed much more toward producing immediately usable results than toward advancing the state of the art; the problem is made worse by the high cost of the talent required to implement system software.

One solution, seen by many quite early in the game, was to make software that could be transported from one processor to another in either of two ways: The machine code could be transported, or high-level language source programs could be transported and recompiled for the new machine.

The first approach works well as long as the machine codes remain constant from processor to processor and the destination processor operates in a manner functionally equivalent to the source processor. The primary drawback is a (perhaps unhappy) marriage to a fixed set of operations and instruction formats. The advantages and drawbacks are exemplified by IBM's 360 and 370 series mainframes. That architecture is now over fifteen years old. Its preservation has allowed the transporting of code from model to model, but it has also inhibited technological growth.

The second approach to portability works well provided that the high-level language facility exists for the destination processor and that the high-level language compilers produce functionally equivalent object modules. The primary disadvantage of this approach is that the required

"It is helpful to think of PL/C as a strictly structured and totally procedural programming language. . . ."

language facility often arrives well after the processor. Also, as a consequence of normal system software development (wherein we keep reinventing the same old wheel), the language processors frequently do not produce functionally equivalent object modules.

The PL/C language has been developed to address the problems inherent in the second approach. PL/C makes possible the implementation of compiler core programs in a processor-independent form, enabling compilers to be developed with much greater speed than before. The PL/C compiler program has itself been implemented in a processor-independent form. Listing One (page 50) is the PL/C listing for the compiler itself. It requires approximately 24 subroutines (a total of about 2K bytes to complete its implementation for any given processor. Listing Two (page 54) is the compiler in macro-assembler code.

PL/C Structural Notation

A PL/C program consists of a sequence of labeled statements, followed by an unlabeled statement consisting of the word "end" (in either upper or lower case), followed by a semicolon. For convenience we will set the end statement aside as a special case and refer to labeled statements simply as "statements."

Statements begin with a label, which may be followed by any number of PL/C specification items, and end with a semicolon. A label consists of a PL/C statement name followed by a colon (or "::<="). A statement name consists of a left angle bracket (<), followed by the name, followed by a right angle bracket (>). The name itself may contain imbedded blanks.

within a PL/C statement as a specification item, there must be a statement labeled with the same name. Obviously, syntax definition must continue until all syntactical elements have been resolved to literal values.

Ordering of statements is arbitrary, except that the first statement must be the highest level of definition used.

PL/C syntax is fully defined in Table I (page 47).

PL/C Program Structure and Flow

It is helpful to think of PL/C as a strictly structured and totally procedural programming language in which the first statement of any program is the main procedure and all subsequent statements are subprocedures. That way, all syntax item specifications within the body of each statement may be regarded as subprocedure calls. The control flow map of any PL/C program, then, is a tree structure at every node of which is a PL/C specification item.

A node may or may not be classified as "output producing." Output may consist of data collected from the input string undergoing compilation, or it may be data generated explicitly by an output specification. The PL/C language provides for the ordered passing of output data up the tree structure from node to node. The output data passed all the way up to the initial node becomes "actual" output: the result of the compilation process. Operationally, then, a compiler programmed in PL/C is not much different from most other compilers. The real differences lie in the development process.

PL/C program flow control is entirely implicit: no language facility exists for explicit transfer of control. Since most programming errors are flow related, this strictly structured approach minimizes many common errors. The problem of attempting to enforce internal structured-programming standards is avoided: not only does the language itself make the use of explicit flow control statements seem unnatural, but it also de-emphasizes the entire concept of "flow" by dispensing with the usual notion that control passes from statement to statement in the order in which the statements appear. In addition, the structured approach makes it easy to partition the development effort and removes most of the difficulties involved in follow-on maintenance and enhancement.

by Morris Dovey

Morris R. Dovey, MRD Systems Incorporated, P. O. Box 147, Spring Valley, MN 55975.

Copyright © 1983 Morris R. Dovey. All rights reserved.

License is granted for personal and non-commercial use of this material. All commercial rights are reserved.

The BNF (Bakus-Naur Format) description of the target language constitutes the bulk of the compiler code. Because published BNF descriptions are readily available for every major programming language in use, the entire scanning logic for a compiler in effect is available on an "off the shelf" basis. This allows the development effort to be concentrated where it belongs: in the generation of correct and efficient compiler output.

Syntactical Elements

The PL/C programming language is a natural extension of the BNF notation for language syntax definition and description. PL/C extends BNF in three essential ways:

1. An output facility has been provided.
2. Input collection and symbol manipulation facilities have been added.
3. A facility has been included to permit in-line inclusion of instructions and/or data for processing at assembly time.

Some changes of notation from BNF are necessary. Character literals in BNF are normally undelimited; in PL/C they are delimited by single quotes in order to deal with the space character.

The asterisk (*), used to denote the optional occurrence of a specification, is placed following the optional item specification. For example:

```
<procedure keyword>::=
'PROC' 'EDURE'*;
```

indicates that the character string "PROC" is required at the current point in the input file being scanned. Once this requirement is satisfied, the string "EDURE" is permitted but not required. Thus the specification "procedure keyword" may be satisfied by either the string "PROC" or the string "PROCEDURE."

The dollar sign (\$) is used to denote the occurrence of a specification one or more times to satisfy the containing specification. An example of its use might be:

```
<integer>::= <decimal digit> $;
```

where the specification "integer" will be satisfied if "decimal digit" is satisfied one or more times.

The dollar sign and asterisk may be combined to indicate that many optional occurrences of the specification are permitted. For example:

```
<skip>::= ' '$*;
```

will cause the compiler to scan to the next nonblank character in the input stream. If the character at the current point in the input stream is not a space character, the scan will not advance; however, the "skip" specification will

OPTIMIZING C86™ controls Charlie...

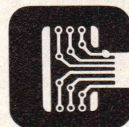
Like a Puppet on a String!



Serious programmers can use Optimizing C86 to control the IBM PC and other MS-DOS/PC-DOS computers. Features include:

- A **Full** and **Standard** implementation of the C language balancing tight control of the machine with the productivity and portability of a high level and standard language.
- **Hardware and operating system interfaces** for: graphics, interrupt control, **8087** use, I/O ports, real time applications and for producing ROMS.
- Use of the standard MS-DOS Linker and a compiler option to produce MASM Assembler output. Helps integrate with MASM and other standard languages like MS FORTRAN and PASCAL.
- A **rich set of libraries** that includes source for K&R functions, extras for string handling, graphics, sorting, floating point (8087 and 8086/88), "**Large**" **model** memory use (1,000K RAM), "**Small**" **memory** model, MS-DOS 1.1, 1.25, 2.0, 2. + + .
- Support for numerous **add-on libraries** including: HALO **Graphics**, C Tools for fundamentals in many areas, PHACT for **ISAM** file management and numerous others. Ask for a list.

Pull Charlie's strings with our fast, complete, proven, reliable C Compiler—the leading compiler for serious programmers of MS-DOS and CPM-86 systems.



Computer Innovations
980 Shrewsbury Avenue
Suite J-506
Tinton Falls, N.J. 07724
(201) 542-5920

**"They Say It All...
We Do It All!"**

Visa and MasterCard accepted.

C86 is a trademark of Computer Innovations, Inc. CPM-86 and MPM-86 are trademarks of Digital Research. MS-DOS is a trademark of Microsoft. PC-DOS is a trademark of International Business Machines.

Circle no. 12 on reader service card.

not fail since the space(s) are only permitted, not required.

The vertical bar (|) is used to indicate that, should the preceding specification fail, the following specification is to be accepted by the compiler as a valid alternative. For example:

```
<arithmetic atom>::=
  <constant> | <variable>;
```

indicates that the specification "arithmetic atom" will be satisfied by the success of either the "constant" specification or the "variable" specification. If the "constant" specification is satisfied, the "variable" specification will not be checked.

All statements are terminated with a semicolon. PL/C programs use a free format, which means that a specification (statement) may extend over as many lines as desired and that several statements may be coded in a single line; in short, line (record) boundaries are ignored.

PL/C Output Facilities

Output specifications in PL/C may occur anywhere within the containing specification and are enclosed in brackets ([]). Three modes of output are available under the current implementation:

Character string output

Hexadecimal value byte output Subordinate node output

Character string output is accomplished by enclosing the string to be generated in single quotes. For example:

```
<finis>::=<skip> 'END' | 'end'
  <skip> ';' ['END'];
```

will bypass blanks, find the keyword "END" in either upper or lower case, bypass any following blanks, and, if a semicolon follows, output the character string "END" (in upper case, as specified).

Numeric output is accomplished by enclosing one or more pairs of hexadecimal digits in angle brackets, as follows:

```
<S370 SVC Op Code>::= [<0A>];
```

Nodal output is accomplished by specification of the subordinate node's position relative to the output specification: where the most recent output-producing node is 1, the next most recent is 2, and so on. In the following example:

```
<pointer>::=<internal label>
  ['DC AL4('1')'];
```

if "internal label" is satisfied, the specification will output the string "DC AL4(" and follow it with the output of the "internal label" specification, concluding with the closing right parenthesis. When

multiple nodal output specifications are used together, they must be separated by commas.

Note that, for node numbering purposes, output specifications are not considered to be *output-producing* specifications. Also, for output purposes, a sequence of specifications separated by vertical bars is considered to be a single node.

Assembler Code Insertion and Built-In Functions

Assembler code may be inserted at any point between specifications by enclosing the code in double angle brackets, as follows:

```
<< Title 'PL/C [V2.0]
  COMPILER MAINLINE' >>
```

An assembler instruction may be inserted at any point between specifications in the containing statement by coding an ampersand (&) followed by the assembler op code mnemonic or macro name. If operands are used, they are enclosed in parentheses and must immediately follow the operation mnemonic. For example:

```
<dummy spec>::=
  &COPY(DSNAME) &CSECT;
```

A minimal set of built-in functions has been implemented for the current PL/C compiler using this facility. For example, a SKIP macro calls an assembler language subroutine to advance the scan pointer to the next nonblank character; this approach speeds up the PL/C compiler considerably. Another example is the MEMB macro, which generates a call to another assembler language subroutine to check the character at the current scan point for membership in the character string supplied as an argument to the macro. Thus:

```
<decimal digit>::=
  '0' | '1' | '2' | '3' | '4' |
  '5' | '6' | '7' | '8' | '9' [1];
```

can be replaced by the less cumbersome and more rapid:

```
<decimal digit>::=
  &MEMB('0123456789') [1];
```

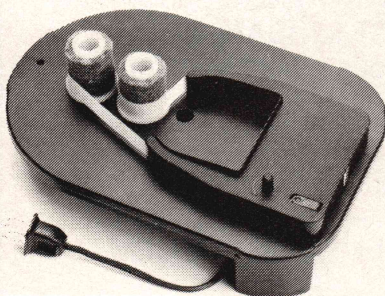
This permits the incorporation of any special facility that may be required either for performance or to meet language requirements (such as attribute analysis for PL/C structure elements). We recommend that this method be reserved for incorporation of new PL/C facilities and that target compiler code be generated using the text inclusion method (angle brackets).

Development History

The PL/C [V1.0] compiler (the initial version) was implemented in Technical Design Labs' Zapple BASIC on a

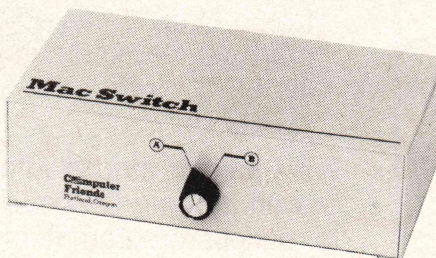
Re-ink any fabric ribbon for less than 5¢. Extremely simple operation. We have a MAC INKER for any printer. Lubricant ink safe for dot matrix printheads. Multi-colored inks, uninked cartridges available. Ask for brochure. Thousands of satisfied customers.

\$54⁹⁵ +



Mac Switch lets you share your computer with any two peripherals (serial or parallel). Ideal for word processors—never type an address twice. Ask us for brochure with tips on how to share two peripherals with MAC SWITCH. Total satisfaction or full refund.

\$99⁰⁰



Order Toll Free 1-800-547-3303

Mac Inker & MacSwitch

**Computer
Friends**

6415 SW Canyon Court
Suite #10
Portland, Oregon 97225
(503) 297-2321

Circle no. 11 on reader service card.

Z80-based microprocessor system. The entire program required approximately 550 BASIC statements and was adequate only to compile the PL/C compiler itself. The PL/C [V2.0] compiler required only 44 PL/C statements, which was fortunate since the PL/C [V1.0] compiler processed approximately 15 statements per hour.

The output of the BASIC version was a string of macro assembler statements, all of which were macro invocations describing common high-level compiler functions. These macro invocations were generated in a form acceptable to most macro assemblers (as were those of subsequent versions), so the PL/C compiler has existed in a processor-independent form from its earliest stages.

A set of approximately two dozen macros was written to generate Z80 microprocessor instructions. Nearly all of the macros were expanded to compiler subroutine calling sequences, and the appropriate set of compiler subroutines was written in Z80 assembler language. These macros and subroutines are the only processor-dependent code involved in implementing the PL/C compiler.

The common subroutine source code file was concatenated to the generated

compiler mainline file, as was the macro definition file. The composite assembler source file was assembled using Technical Design Labs' macro assembler program.

PL/C [V2.0] was up and running after eight iterations of the process just described. This version was functionally equivalent to PL/C [V1.0] and compiled at a rate of approximately two statements per second — a considerable speed improvement over the original BASIC program! At this point, the PL/C source code was rewritten to provide much greater power and flexibility, additional built-in functions were brought into play, and existing functions were made more flexible.

In generating PL/C [V3.0], which produces Z80 BASIC assembler source code, the macro definitions were maintained in a separate file and were brought into play via an 'INSERT' statement generated as part of a prolog generated by PL/C [V2.1]. The common subroutines were maintained in a separate relocatable object library and were not made a part of the object compiler until link-edit time. This approach permitted separation of compiler components into segments that were much easier to maintain.

PL/C [V3.0] was judged adequate for generation of additional compilers; with this version, we considered the PL/C programming language implementation to be complete. Additional PL/C compiler programming activity is categorized as either maintenance or enhancement.

Availability of PL/C

Readers may obtain a personal-use (non-commercial) PL/C license and Z80-executable versions of PL/C Compilers which generate TDL Assembler and Macro code, the PL/C Subroutine Library in relocatable object form, and PL/C User Manual for \$25. On receipt of payment I will send a license agreement to be signed and returned. On receipt of the license agreement I will mail the diskette. Commented source listings of the PL/C subroutine library (on diskette) and commercial licenses are also available.

DDJ

(Listings begin on page 50)

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 193.

Table I.

```

<program>::= <statement>$ <end>;

<statement>::= <generate> | <compiler statement>;

<generate>::= ' '$# '<<' &NEXT('>>');

<compiler statement>::= <statement label> <statement body>;

<statement label>::= <compiler label> ' '$# ':' ':'=';

<compiler label>::= ' '$# '<' <name root> <name segment>$#
                    ' '$# '>';

<name root>::= ' '$# <letter> <alphanumeric>$#;

<letter>::= <lower case> | <upper case>;

<lower case>::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' |
                'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' |
                'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' |
                'y' | 'z';

<upper case>::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' |

```

(Continued on next page)


```

'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' |
'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' |
'Y' | 'Z';

<alphanumeric>::= <letter> | <digit>;

<digit>::= '0' | '1' | '2' | '3' | '4' |
'5' | '6' | '7' | '8' | '9';

<name segment>::= ' ' '$* <alphanumeric>$';

<statement body>::= <action spec> <semi> | <statement body>;

<action spec>::= <generate> | <scan item> | <output item>;

<scan item>::= <scan call> | <macro call> | <scan string> |
<scan value> | <parenthetic group>;

<scan call>::= <compiler label> <modifier>*;

<macro call>::= ' '$* '&' <built in> | <user macro>;

<user macro>::= <name root> <macro parameter list>*;

<macro parameter list>::= ' '$* '(' <macro arg>
<addl macro arg>$* ' '$* ')';

<macro arg>::= <string> | <macro value> | <name root>;

<string>::= ' '$* ''' <string character>$ ''';

<string character>::= <alphanumeric> | <special>;

<special>::= ' ' | '!' | '"' | '#' | '$' | '%' | '&' | '(' |
')' | '*' | ':' | '=' | '-' | '{' | '}' | '[' |
']' | '~' | '^' | '+' | ';' | '@' | '|' | '\' |
' ' | '<' | '>' | ',' | '.' | '?' | '/' | ' ' ;

<macro value>::= ' '$* <digit>$;

<addl macro arg>::= ' '$* ',' <macro arg>;

<scan string>::= <string> <modifier>*;

<scan value>::= ' '$* '#' ''' <hex pair>$ ' '$* '''
<modifier>*;

<hex pair>::= ' '$* <hex digit> <hex digit>;

```



```

<hex digit>::= <digit> | 'A' | 'B' | 'C' | 'D' | 'E' | 'F';

<parenthetic group>::= ' '$* '(' <action spec>$ ' '$* ') '

                        <modifier>*;

<modifier>::= ' '$* <optionally many> | <one or more> |

                        <optional> <alt>*;

<optionally many>::= '$' ' '$* '*';

<one or more>::= '$';

<optional>::= '*';

<alt>::= ' '$* '|' <scan item>;

<output item>::= ' '$* '[' <output spec>$ ' '$* ']' ;

<output spec>::= <node spec> | <output string> | <output value>;

<node spec>::= <node number> ( ' '$* ',' <node number> )$*;

<node number>::= ' '$* <digit>$;

<output string>::= <string>;

<output value>::= ' '$* '<' ' '$* <hex pair>$ ' '$* '>';

<semi>::= ' '$* ';' ;

<end>::= ' '$* 'END' | 'end' <semi>;

end;

```

(Listing begin on page 50)

NOW FOR THE
IBM PC
+ COMPATIBLES



WINDOWS FOR C™

SCREEN MANAGEMENT TOOL FOR C PROGRAMMERS

A COMPLETE WINDOW DISPLAY SYSTEM

- Unlimited windows and text files
 - Instant screen changes
 - Horizontal and vertical scrolling
 - Variable text margins
 - Word wrap option
 - ASCII file handling routines
(including tab expansion)
- BAR GRAPH ROUTINES INCLUDED

Available For: Lattice C, C 86, Microsoft C, DeSmet C

FOR ALL DISPLAY TASKS

FAST

Line-oriented ASM subroutines

COMPREHENSIVE

Access to all text mode capabilities

VERSATILE

Building block subroutines

FULLY DOCUMENTED

Introductory Special \$119.95
(\$150 after May 31)
Demo disk & manual \$30
(Applies toward purchase)

A PROFESSIONAL SOFTWARE TOOL FROM
CREATIVE SOLUTIONS
21 Elm Ave., Box D3 Richford, VT 05476

For Order or Information: 802-848-3804
Master Card & Visa Accepted
Shipping \$2.50
VT residents add 4% tax

Circle no. 14 on reader service card.

PL/C Compiler Listing (Text begins on page 44)

Listing One

```
<<.TITLE      'PL/C [2.2] Programming Language for Compilers'
.SBTTL 'Copyright (C) 1981 - MRD Systems, Inc., POB 147, Spring Valley, MN'
.MAIN.:CALL    $INIT#           ; Initialize Compiler
>>
<program>: <statement>$ <end> [2,1];

<statement>: <gen block> | <compiler statement>;

<gen block>: <generate> [1] &FLSH;

<generate>: &SKIP '<<' &COLL(ON) &NEXT('>>') &COLL(OFF) [1];

<compiler statement>: <statement label> <statement body>
                      [2,1] RTRN
                      ']' &FLSH;

<statement label>: <compiler label> &SKIP ':' ':= '* &SYMB(3) ['
; '4'
'1':'];

<compiler label>: &SKIP '<' <label root> <label segment>$* &SKIP '>' [3,2];

<label root>: &SKIP &COLL(ON) <letter> <alphanumeric>$* &COLL(OFF) [2,1];

<letter>: <lower case> | <upper case> [1];

<lower case>: &MEMB('abcdefghijklmnopqrstuvwxyz') [1];

<upper case>: &MEMB('ABCDEFGHIJKLMNOPQRSTUVWXYZ') [1];

<alphanumeric>: <letter> | <digit> [1];

<digit>: &MEMB('0123456789') [1];

<label segment>: ' ' &SKIP &COLL(ON) <alphanumeric>$ &COLL(OFF) [' '1];

<statement body>: <action specification> <semi> | <statement body> [2,1];

<action specification>: <generate> | <scan item> | <output item> [1];

<scan item>: <scan call> | <macro call> | <scan string> | <scan value> |
              <parenthetic group> [1];

<scan call>: <compiler label> <modifier>* &SYMB(2)
              [' XFER '1','2' ; '3'
              '];

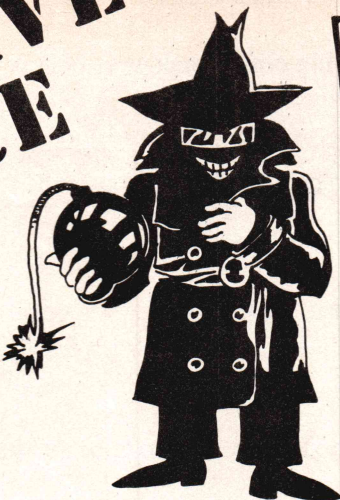
<macro call>: &SKIP '&' <built in> | <user macro> [1];

<built in>: <next> | <member> | <collect> [1];
```

(Continued on page 52)

SUBVERSIVE SOFTWARE

*So cheap and useful
it's...dangerous!*



A radical idea in software development.
Useful Pascal programs, with:

- Complete annotated source listings;
- Disk(s) with source code and compiled code;
- User manual;
- Complete programmer documentation describing data structures and algorithms; and giving suggestions for modification.

Modify them, include them in your own systems, or simply use them.

A growing library of software tools you'll find hard to resist.

SUBVERSIVE SOFTWARE

TPL

The Text Processing Language. A text-file runoff program consisting of a set of text-processing primitive commands from which more complex commands (macros) can be built (as in Logo). Features include:

- Complete customization of text processing through macro definition and expansion, looping structures, and conditional statements;
- Adapts to any printer;
- Pagination;
- Text justification and centering;
- Indexing and tables of contents;
- Superscripts and subscripts;
- Bolding and underlining;
- Multiple headers and footers;
- End notes and footnotes;
- Widow and orphan suppression;
- Floating tables and 'keeps.'

\$50

SUBVERSIVE SOFTWARE

DBX

Blocked Keyed Data Access Module. Maintains disk files of keyed data. Can be used for bibliographies, glossaries, multi-key data base construction, and many other applications.

- Variable-length keys;
- Variable-length data;
- Sequential access and rapid keyed access;
- Single disk access per operation (store, find, delete) in most cases;
- Multiple files;
- Dynamic memory allocation for RAM-resident index and current "page" of entries;
- Includes demonstration program and testbed program.

\$50

SUBVERSIVE SOFTWARE

PDMS

The Pascal Data Management System. A user-oriented data management system in which numeric and alphanumeric data are stored in tables with named columns and numbered rows. Currently being used for dozens of different kinds of business and scientific applications, from inventory management to laboratory data analysis. Includes over 20 Pascal programs; more than 10,000 lines of code. Main features include:

- Maximum of 32,767 rows per file;
- Maximum of 400 characters per row, and 40 columns per table;
- Full-screen editing of rows and columns, with scrolling, windowing, global search/replace, and other editing features;
- Sorting, copying, merging, and reducing routines;
- Mailing label program;
- Reporting program generates reports with control breaks, totals and subtotals, and selects rows by field value; many other reporting features;
- Cross-tabulation, correlations, and multiple regression;
- Video-display-handling module;
- Disk-file-handling module.

Many other features. UCSD formats only.

\$250

SUBVERSIVE SOFTWARE

ZED

Full-screen text editor; designed to be used either with TPL or by itself.

- Full cursor control;
- Insert mode with word wrap;
- 'Paint' mode;
- Single-keystroke or dual-keystroke commands;
- Command synonyms;
- Global search and replace;
- Block move, block copy, and block delete.

\$50

SUBVERSIVE SOFTWARE

SCINTILLA

A log logit curve fitting program for radio-immunologic data; must be used with PDMS (described above).

- Multiple protocol files;
- Quality control files;
- Four-parameter non-linear curve fit.

UCSD formats only.

\$250

SUBVERSIVE SOFTWARE

CHROME

Chromatography data analysis program:

- Graphic display of analog data;
- Panning and zooming;
- Automatic peak-finding and baseline calculation;
- Full interactive peak editing;
- Computation of peak areas;
- Strip charts on C. Itoh and EPSON printers.

\$100

SUBVERSIVE SOFTWARE

PLANE

Planimetry program:

- Bit-pad entry of cross sections;
- Real-time turtlegraphics display;
- Calculation of areas;
- Saves calculations to text file.

\$100

SUBVERSIVE SOFTWARE

MINT

A terminal emulation program for communication between computers of any size.

- User-configurable uploading and downloading of files;
- X-ON/X-OFF and EOB/ACK protocols;
- Interrupt-driven serial input (for Prometheus Versacard in Apple II);
- Printer-logging.

\$50

For more information, call 919-942-1411. To order, use form below or call our toll-free number: 1-800-XPASCAL

Check appropriate boxes:

FORMAT

- ☐ 8" UCSD SSSD
- ☐ 5 1/4" Apple Pascal
- ☐ 5 1/4" UCSD IBM PC 320k
- ☐ 8" CP/M SSSD
- ☐ 5 1/4" IBM MS-DOS
- ☐ 5 1/4" CP/M Osborne

PRODUCT

- ☐ DBX
- ☐ PDMS
- ☐ TPL
- ☐ ZED
- ☐ MINT
- ☐ SCINTILLA
- ☐ CHROME
- ☐ PLANE

PRICE

- \$ 50
- \$250
- \$ 50
- \$ 50
- \$ 50
- \$250
- \$100
- \$100

Name _____

Address _____

☐ MasterCard

☐ VISA

☐ Check

☐ C.O.D.

(Please include card # and expiration date)



SUBVERSIVE SOFTWARE

A division of Pascal & Associates,

135 East Rosemary St., Chapel Hill, NC 27514

Apple and Apple Pascal are trademarks of the APPLE Computer Corp. IBM and IBM PC are trademarks of International Business Machines. UCSD Pascal is a trademark of the Regents of the University of California. Osborne is a trademark of Osborne Computer. EPSON is a trademark of EPSON America, Inc. C. Itoh is a trademark of C. Itoh Electronics.

Circle no. 42 on reader service card.

PL/C Compiler Listing (Listing continued, text begins on page 44)

Listing One

```
<next>: 'NEXT' | 'next' &SKIP '(' <string> &SKIP ')' <modifier>*
      ['      NEXT      '3','1'
];

<member>: 'MEMB' | 'memb' &SKIP '(' <string> &SKIP ')' <modifier>*
      ['      MEMB      '3','1'
];

<collect>: 'COLL' | 'coll' &SKIP '(' <col on> | <col off> &SKIP ')'
      ['      COLL      '2'
];

<col on>: 'ON' | 'on' ['ON'];

<col off>: 'OFF' | 'off' ['OFF'];

<user macro>: <label root> <macro parameter list>* ['      '2,'1'
];

<macro parameter list>: &SKIP '(' <macro arg> <addl macro arg>$* &SKIP ')'
      ['      '3,2];

<macro arg>: <string> | <macro value> | <label root> [1];

<string>: &SKIP ''' &COLL(ON) <string character>$ &COLL(OFF) ''' [<60>2<60>];

<string character>: <alphanumeric> | <special> | <string quote> [1];

<special>: &MEMB(' !"#$$%&()*:=-{[]~^+;@|\ <, > . ? /
      ') [1];

<string quote>: &COLL(OFF) '''' &COLL(ON) ['''];

<macro value>: &SKIP &COLL(ON) <digit>$ &COLL(OFF) [1];

<addl macro arg>: &SKIP '.' <macro arg> ['.'1];

<scan string>: <string> <modifier>* ['      SCAN      '2,'1'
];

<scan value>: &SKIP '#' ''' <hex pair>$ &SKIP ''' <modifier>*
      ['      SCNR      '<60>3<60>', '1'
];

<hex pair>: &SKIP &COLL(ON) <hex digit> <hex digit> &COLL(OFF) [2,1];

<hex digit>: &MEMB('0123456789ABCDEF') [1];

<parenthetic group>: &SKIP '(' <action specification>$ &SKIP ')' <modifier>*
      &LGEN &LGEN ['      XFER      '2,'3'
      GOTO      '1'
      '2': '5' RTRN
      '1': ];
```


<parenthetic body>: <action specification> &SKIP ')' | <parenthetic body> [3,1];

<modifier>: &SKIP <alt mod> | <optionally many> | <one or more> |
<optional> | <default zero> [1];

<alt mod>: <multi alt> | <single alt> [1];

<multi alt>: '\$' &SKIP '|' ['6'];

<single alt>: '|' ['4'];

<optionally many>: '\$' &SKIP '*' ['3'];

<one or more>: '\$' ['2'];

<optional>: '*' ['1'];

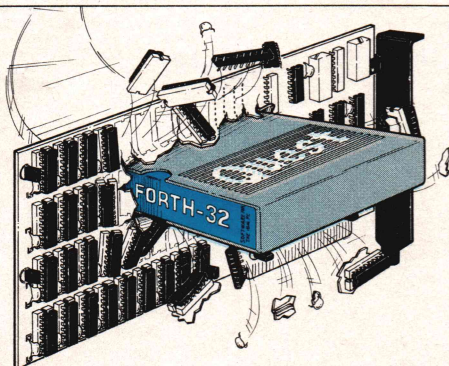
<default zero>: ['0'];

<output item>: &SKIP '[' <output spec>\$ &SKIP '[' [2];

<output spec>: <node spec> | <output string> | <output value> [1];

<node spec>: <node number> <addl node number>\$* [2,1];

(Continued on next page)



Break Through the 64K Barrier!

FORTH-32™ lets you use up to one megabyte of memory for programming. A Complete Development System! Fully Compatible Software and 8087 Floating Point Extensions.

Quest™
Quest Research, Inc.

303 Williams Ave.
Huntsville, AL 35801
(205) 533-9405

Call today toll-free or
contact a participating
Computerland store.

800-558-8088

Now available for the IBM PC, PC-XT, COMPAQ, COLUMBIA MPC,
and other PC compatibles!

IBM, COMPAQ, MPC, and FORTH-32 are trademarks of IBM, COMPAQ, Columbia Data Products, and Quest Research, respectively.

Circle no. 47 on reader service card.



Menu Driven Programs running under SPELLBINDER

Dear John 

will free your operators from entry errors. Completely menu driven Mailing List Management Programs with thoughtfully formatted screens to ease the entry of addresses and names. Integrity of the data file created is automatically verified. No more printing 'bad' mailing labels with misplaced data; AND each address record can be individually classified to allow inclusion (or exclusion) when printing labels or form letters.

(10 Programs) **\$89.50**

SBMenu SBHelp

make your version of Spellbinder menu driven. No more fumbling through manuals for obscure or little-used commands. The complete instruction set can be displayed at any time without disturbing the text on the screen. These two programs put Spellbinder into the world class of word processors, allowing the user to easily use its truly outstanding features.

(With two help files) **\$55.00**

AutoSave

automatically saves the text on the screen to a previously named file. Prevents the inadvertent but all so common save to a wrong name. Works with source code and program code as well as with text files. All instructions on disk.

(3 Programs) **\$37.50**

Send Check, Money Order, Visa, Master Charge:

COMPUTER RESOURCES OF WAIMEA
P.O. Box 1206
Kamuela, Hawaii 96743
Phone: (808) 885-7905

Circle no. 13 on reader service card.

PL/C Compiler Listing (Listing continued, text begins on page 44)

Listing One

```
<node number>: &SKIP &COLL(ON) <digit>$ &COLL(OFF)
      ['      NODE      '1'
'];

<addl node number>: &SKIP ',' <node number> [1];

<output string>: <string>
      ['      STOP      '1'
'];

<output value>: &SKIP '<' &SKIP <hex pair>$ &SKIP '>'
      ['      OHEX      '2'
'];

<semi>: &SKIP ';;';

<end>: &SKIP 'END' | 'end' <semi>
      ['      .END
'<1A>];

end;
```

End Listing One

Listing Two

```
.TITLE 'PL/C [2.2] Programming Language for Compilers'
.SBTTL 'Copyright (C) 1981 - MRD Systems, Inc., POB 147, Spring Valley, MN'
.MAIN::CALL $INIT# ; Initialize Compiler

; program

X0001: XFER X0002,2 ; statement
      XFER X0003.0 ; end
      NODE 2
      NODE 1
      RTRN

; statement

X0002: XFER X0004,4 ; gen block
      XFER X0005.0 ; compiler statement
      RTRN

; gen block

X0004: XFER X0006,0 ; generate
      NODE 1
      FLSH
      RTRN

; generate

X0006: SKIP
      SCAN '<<',0
      COLL ON
```

(Continued on page 56)

Z80 Software

SOFTWARE DESCRIPTIONS

TPM (TPM I) - \$80 A Z80 only operating system which is capable of running CP/M programs. Includes many features not found in CP/M such as independent disk directory partitioning for up to 255 user partitions, space, time and version commands, date and time, create FCB, chain program, direct disk I/O, abbreviated commands and more! Available for North Star (either single or double density), TRS-80 Model I (offset 4200H) or II, Versafloppy I, or Tarbell I.

TPM-II - \$125 An expanded version of TPM which is fully CP/M 2.2 compatible but still retains the extra features our customers have come to depend on. This version is super FAST. Extended density capability allows over 600K per side on an 8" disk. Available preconfigured for Versafloppy II (8" or 5"), Epson QX-10, Osborne II or TRS-80 Model II.

CONFIGURATOR I

This package provides all the necessary programs for customizing TPM for a floppy controller which we do not support. We suggest ordering this on single density (8SD).

Includes: TPM-II (\$125), Sample BIOS (BIOS) SOURCE (\$FREE), MACRO II (\$100), LINKER (\$80), DEBUG I (\$80), QED (\$150), ZEDIT (\$50), TOP I (\$80), BASIC I (\$50) and BASIC II (\$100)

\$815 Value

NOW \$250

CONFIGURATOR II

Includes: TPM-II (\$125), Sample BIOS (BIOS) SOURCE (\$FREE), MACRO II (\$100), MACRO III (\$150), LINKER (\$80), DEBUG I (\$80), DEBUG II (\$100), QSAL (\$200), QED (\$150), ZTEL (\$80), TOP II (\$100), BUSINESS BASIC (\$200) and MODEM SOURCE (\$40) and DISASSEMBLER (\$80)

\$1485 Value

NOW \$400

MODEL I PROGRAMMER

This package is only for the TRS-80 Model I. Note: These are the ONLY CDL programs available for the Model I. It includes: TPM I (\$80), BUSINESS BASIC (\$200), MACRO I (\$80), DEBUG I (\$80), ZDDT (\$40), ZTEL (\$80), TOP I (\$80) and MODEM (\$40)

\$680 Value

NOW \$175

MODEL II PROGRAMMER

This package is only for the TRS-80 Model II. It includes: TPM-II (\$125), BUSINESS BASIC (\$200), MACRO II (\$100), MACRO III (\$150), LINKER (\$80), DEBUG I (\$80), DEBUG II (\$100), QED (\$150), ZTEL (\$80), TOP II (\$100), ZDDT (\$40), ZAPPLE SOURCE (\$80), MODEM (\$40), MODEM SOURCE (\$40) and DISASSEMBLER (\$80)

\$1445 Value

NOW \$375

BASIC I - \$50, a 12K+ basic interpreter with 7 digit precision.

BASIC II - \$100, A 12 digit precision version of Basic I.

BUSINESS BASIC - \$200, A full disk extended basic with random or sequential disk file handling and 12 digit precision (even for TRIG functions). Also includes PRIVACY command to protect source code, fixed and variable record lengths, simultaneous access to multiple disk files, global editing, and more!

ACCOUNTING PACKAGE - \$300, Written in Business Basic. Includes General Ledger, Accounts Receivable/Payable, and Payroll. Set up for Hazeltine 1500 terminal. Minor modifications needed for other terminals. Provided in unprotected source form.

MACRO I - \$80, A Z80/8080 assembler which uses CDL/TDL mnemonics. Handles MACROs and generates relocatable code. Includes 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical ops, local and global symbols, linkable module generation, and more!

MACRO II - \$100, An improved version of Macro I with expanded linking capabilities and more listing options. Also internal code has been greatly improved for faster more reliable operation.

MACRO III - \$150, An enhanced version of Macro II. Internal buffers have been increased to achieve a significant improvement in speed of assembly. Additional features include line numbers, cross reference, compressed PRN files, form feeds, page parity, additional pseudo-ops, internal setting of time and date, and expanded assembly-time data entry.

DEVELOPER I

Includes: MACRO I (\$80), DEBUG I (\$80), ZEDIT (\$50), TOP I (\$80), BASIC I (\$50) and BASIC II (\$100)

\$440 Value

NOW \$150

DEVELOPER II

Includes: MACRO II (\$100), MACRO III (\$150), LINKER (\$80), DEBUG I (\$80), DEBUG II (\$100), BUSINESS BASIC (\$200), QED (\$150), TOP II (\$100), ZDDT (\$40), ZAPPLE SOURCE (\$80), MODEM SOURCE (\$40), ZTEL (\$80), and DISASSEMBLER (\$80).

\$1280 Value

NOW \$350

DEVELOPER III

Includes: QSAL (\$200), QED (\$150), BUSINESS BASIC (\$200), ZTEL (\$80) and TOP II (\$100)

\$730 Value

NOW \$300

COMBO

Includes: DEVELOPER II (\$1280), ACCOUNTING PACKAGE (\$300), QSAL (\$200) and 6502X (\$150)

\$1930 Value

NOW \$500

LINKER - \$80, A linking loader for handling the linkable modules created by the above assemblers.

DEBUG I - \$80, A tool for debugging Z80 or 8080 code. Disassembles to CDL/TDL mnemonics compatible with above assemblers. Traces code even through ROM. Commands include Calculate, Display, Examine, Fill, Goto, List, Mode, Open File, Put, Set Wait, Trace, and Search.

DEBUG II - \$100, A superset of Debug I. Adds Instruction Interpreter, Radix change, Set Trap/Conditional display, Trace options, and Zap FCB

6502X - \$150, A 6502 cross assembler. Runs on the Z80 but assembles 6502 instructions into 6502 object code! Similar features as our Macro assemblers.

QSAL - \$200, A SUPER FAST Z80 assembler. Up to 10 times faster than conventional assemblers. Directly generates code into memory in one pass but also to offset for execution in its own memory space. Pascal like structures: repeat, until, if, then, else, while, do, begin, end, case. Of Multiple statements per line, special register handling expressions, long symbol names, auto and modular assembly, and more! This one uses ZILOG Mnemonics.

QED - \$150, A screen editor which is both FAST and easy to learn. Commands include block delete, copy, and move to a named file or within text, repeat previous command, change, locate, find at start of line, and numerous cursor and window movement functions. Works with any CRT having clear screen, addressable cursor, clear to end of line, clear to end of screen, and 80X24.

DISK FORMATS

When ordering software specify which disk format you would like.

CODE	DESCRIPTION
8SD	8" IBM 3740 Single Density (128 bytes/26 sectors/77 tracks)
8DD	8" Double Density (256 bytes/26 sectors/77 tracks)
8XD	8" CDL Extended Density (1024 bytes/8 sector/77 tracks - 616K)
5SD	5.25" Single Density (TRS80 Model I, Versafloppy I, Tarbell I)
5EP	5.25" Epson Double Density
5PC	5.25" IBM PC Double Density
5XE	5.25" Xerox 820 Single Density
5OS	5.25" Osborne Single Density
5ZA	5.25" Z80 Apple (Softcard compatible)

TPM INFO When ordering TPM I or II, in addition to Disk Format, please specify one of the following codes:

TPM I:	DESCRIPTION
NSSD/H	North Star Single Density for Horizon I/O
NSSD/Z	North Star Single Density for Zapple I/O
NSDD/H	North Star Double Density for Horizon I/O
NSDD/Z	North Star Double Density for Zapple I/O
TRS80-I	TRS-80 Model I (4200H Offset)
TRS80II	TRS-80 Model II
V18	Versafloppy I 8"
V15	Versafloppy I 5.25"
V118	Versafloppy II 8" (XD)
V115	Versafloppy II 5.25"
TRS80II	TRS-80 Model II (XD)

Prices and Specifications subject to change without notice.
TPM, Z80, CP/M, TRS80 are trademarks of CDL, Zilog, DRI and Tandy respectively.

ZTEL - \$80, An extensive text editing language and editor modelled after DEC's TECO

ZEDIT - \$50, A mini text editor. Character/line oriented. Works well with hardcopy terminals and is easy to use. Includes macro command capability.

TOP I - \$80, A Text Output Processor for formatting manuals, documents, etc. Interprets commands which are entered into the text by an editor. Commands include justify, page number, heading, subheading, centering, and more

TOP II - \$100, A superset of TOP I. Adds: embedded control characters in the file, page at a time printing, selected portion printing, include/merge files, form feed/CRLF option for paging, instant start up, and final page ejection.

ZDDT - \$40, This is the disk version of our famous Zapple monitor. It will also load hex and relocatable files.

ZAPPLE SOURCE - \$80, This is the source to the SMB ROM version of our famous Zapple monitor. It can be used to create your own custom version or as an example of the features of our assemblers. Must be assembled using one of our assemblers.

MODEM - A communication program for file transfer between systems or using a system as a terminal. Based on the user group menu but modified to work with our SMB board or TRS-80 Models I or II. You must specify which version you want.

MODEM SOURCE - \$40, For making your own custom version. Requires one of our Macro Assemblers.

DISASSEMBLER - \$80, Does bulk disassembly of object files creating source files which can be assembled by one of our assemblers.

HARDWARE

S-100 — **SMB II Bare Board \$50**, "System Monitor Board" for S-100 systems, 2 serial ports, 2 parallel ports, cassette interface, 4K memory (ROM, 2708 EPROM, 2114 RAM), and power on jump. When used with Zapple ROM below, it makes putting a S-100 system together a snap.

Zapple ROM \$35, Properly initializes SMB I/II hardware, provides a powerful debug monitor.

IBM PC — **Big Blue Z80 board \$595**, Add Z80 capability to your IBM Personal Computer. Runs CP/M programs but does not require CP/M or TPM. Complete with Z80 CPU, 64K add on memory, serial port, parallel port, time and date clock with battery backup, hard disk interface, and software to attach to PC DOS and transfer programs. Mfr'd by QCS.

50% Discount on all CDL software ordered at the same time as a Big Blue (and for the Big Blue).

APPLE II — **Chairman Z80 \$345**, Add Z80 capability to your Apple II/II Plus computer. Runs CP/M programs with our more powerful TPM. Includes 64K memory add on (unlike the competition this is also useable by the 6502/DOS as well as the Z80). TPM, QSAL assembler, QED Screen Editor, and Business Basic. Mfr'd by AMT Research.

Apple Special \$175, Buy the Apple Z80 Developer at the same time as the 'Chairman' and pay only \$175 instead of \$325.

APPLE Z80 DEVELOPER

Includes: 6502X (\$150), MACRO II (\$100), MACRO III (\$150), QSAL (\$200), QED (\$150), LINKER (\$80), DEBUG I (\$80), DEBUG II (\$100), ZDDT (\$40) and BUSINESS BASIC (\$200)

VALUE: \$1250

NOW \$325

\$175 when purchased with AMT "Chairman" Board

ORDERING INFORMATION:

VISA/MasterCard/C.O.D.
Call or Write With Ordering Information....



OEMS:

Many CDL products are available for licensing to OEM's. Write to Carl Galletti with your requirements.

Dealer Inquiries Invited.

For Phone Orders ONLY Call Toll Free...

1-(800) 458-3491

(Except Pa.)

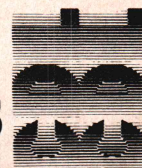
Ask For Extension #15

For information and Tech Queries call
(609) 599-2146

Computer Design Labs

342 Columbus Avenue/Trenton, NJ 08629

Circle no. 10 on reader service card.



Advertise During



Months

May - June - July

June Space Reservations Deadline:

April 5, '84

Material Deadline:

April 12, '84

Contact:

Walter Andrzejewski

Beatrice Blatteis

Alice Hinton

Dr. Dobb's Journal

P.O. Box E

Menlo Park, CA 94026

Get the power of your Z80, and the elegance
of direct access to CP/M functions
from your high level programs with

SYNLIB utility library

SYNLIB consists of MICROSOFT compatible object
code that may be called from any high level language
that uses MICROSOFT parameter passing conventions.

SYNLIB gives you extremely powerful array and buffer
manipulation using the Z80 LDIR instruction; program
access to the CP/M CCP console command line; high
speed disk block I/O; a high quality random number
generator; hex to ASCII conversion optimized by special
Z80 instructions; program chaining; and more.

And, because our programmer abhors a vacuum, each 8"
floppy comes packed with some of the most valuable
public domain software, including available source, ab-
solutely free. You get **SWEEP**, a menued disk utility that
makes a computer phobe a systems programmer;
UNSPPOOL, so you can print and use your computer
without buying an expensive buffer; **I**, to get multiple
commands on a line; **MODEM7**, so that you too can join
the free software movement; and many others.

SYNLIB \$50.00 8" SSSD CP/M format

SOURCE: \$100.00

Licensing for commercial use available.

SYNTAX CONSTRUCTS, Inc.

14522 Hiram Clarke, Houston, Texas 77045

(713) 434-2098

CP/M is a registered trademark of Digital Research, Inc.
Microsoft is a registered trademark of Microsoft Corp.

Circle no. 63 on reader service card.

if you use CP/M[®] then you need DISK FIX!

DISK FIX is a disk editor which can
display, edit or copy any sector of a
CP/M floppy and/or hard disk. The
DISK FIX utility can be used to recover
files from disks with damaged direc-
tories, to reconstruct files with bad sec-
tors, to restore erased files and to do
general disk editing.

DISK FIX automatically configures to
floppy and hard disks, just insert the
program disk and it is ready to run. A
single CPU license is available for \$150.

Call our software HOT LINE

906/228-7622.

The Software Store

706 Chippewa Square • Marquette MI 49855

Circle no. 58 on reader service card.

PL/C Compiler Listing

Listing Two (Listing continued, text begins on page 44)

```
NEXT      `>>>`.0
COLL      OFF
NODE      1
RTRN
```

; compiler statement

```
X0005:  XFER      X0007,0      ; statement label
        XFER      X0008,0      ; statement body
        NODE      2
        NODE      1
        STOP      `
```

RTRN

```
FLSH
RTRN
```

; statement label

```
X0007:  XFER      X0009,0      ; compiler label
        SKIP
        SCAN      `:`.0
        SCAN      `:=`,1
        SYMB      3
        STOP      `
```

; `

```
NODE      4
STOP      `
```

; `

```
NODE      1
STOP      `:`.
RTRN
```

; compiler label

```
X0009:  SKIP
        SCAN      `<`.0
        XFER      X000A,0      ; label root
        XFER      X000B,3      ; label segment
        SKIP
        SCAN      `>`.0
        NODE      3
        NODE      2
        RTRN
```

; label root

```
X000A:  SKIP
        COLL      ON
        XFER      X000C,0      ; letter
        XFER      X000D,3      ; alphanumeric
        COLL      OFF
        NODE      2
        NODE      1
        RTRN
```


; letter

```
X000C:  XFER    X000E,4      ; lower case
         XFER    X000F,0      ; upper case
         NODE    1
         RTRN
```

; lower case

```
X000E:  MEMB    `abcdefghijklmnopqrstuvwxyz`,0
         NODE    1
         RTRN
```

; upper case

```
X000F:  MEMB    `ABCDEFGHIJKLMNOPQRSTUVWXYZ`,0
         NODE    1
         RTRN
```

; alphanumeric

```
X000D:  XFER    X000C,4      ; letter
         XFER    X0010,0     ; digit
         NODE    1
         RTRN
```

; digit

```
X0010:  MEMB    `0123456789`.0
         NODE    1
         RTRN
```

; label segment

```
X000B:  SCAN    ``.0
         SKIP
         COLL    ON
         XFER    X000D,2      ; alphanumeric
         COLL    OFF
         STOP    ``
         NODE    1
         RTRN
```

; statement body

```
X0008:  XFER    X0011.0      ; action specification
         XFER    X0012.4      ; semi
         XFER    X0008,0      ; statement body
         NODE    2
         NODE    1
         RTRN
```

; action specification

```
X0011:  XFER    X0006,4      ; generate
         XFER    X0013.4      ; scan item
         XFER    X0014.0      ; output item
         NODE    1
         RTRN
```

(Continued on next page)

ICs PROMPT DELIVERY!!!

SAME DAY SHIPPING (USUALLY)

DYNAMIC RAM

256K	150 ns	\$85.00
64K	200 ns	5.97
64K	150 ns	6.09
64K	120 ns	6.90
16K	200 ns	1.56

EPROM

27128	300 ns	\$22.50
2764	250 ns	8.30
2732	450 ns	4.75
2716	450 ns	3.60
2532	450 ns	4.75

STATIC RAM

6264P-15	150 ns	\$35.97
6264LP-15	150 ns	42.00
6116P-3	150 ns	5.75

MasterCard VISA or UPS CASH COD

Factory New, Prime Parts

MICROPROCESSORS UNLIMITED

24 000 South Peoria Ave. (918) 267-4961

BEGGS, OK. 74421

Prices shown above are for Jan. 13, 1984.

Please call for current & volume prices. Prices subject to change. Please expect higher prices on some parts due to world wide shortages. Shipping and insurance extra. Cash discount prices shown. Federal Express Standard Air is \$5.99. Orders received by 6 PM CST can be delivered to you by the next morning.

Circle no. 35 on reader service card.



NEW version for 1984!

- Enables programs written for Digital Research CP/M 2.2 to run under Cromemco CDOS and vice versa. **INTRCEPT** release 3 automatically recognizes the host system, and emulates CP/M 2.2 if the host is CDOS, or emulates CDOS if the host is CP/M 2.2.
- No programming, delivered ready-to-run.
- Customizable...comes with CDOS emulator source, CP/M 2.2 emulator source optional.
- Z80 assembly language, no program or operating system modifications.

\$150 w/CDOS emulator source

\$250 w/CDOS & CP/M emulator source

8" SSD, inquire about 5 1/4"
add \$3 shipping, add 6% tax in CA
VISA, MC, check



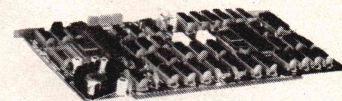
microSystems

16809 Sagewood Lane
Poway, California 92064
(619) 693-1022

Circle no. 45 on reader service card.

80 CHARACTER VIDEO BOARD

- **WORDSTAR/dBASE II OPTION**
- **TYPE AHEAD KEYBOARD BUFFER**



- 25 LINE NON-SCROLL OPTION
- Z80 CPU and 8275 CRTC S-100
- CHARACTER GRAPHICS
- ADAPTABLE SOFTWARE
- ORDER ASSEMBLED & TESTED OR PRE-SOLDERED (ADD YOUR IC's)

VDB - A2 bare board from \$49.50

Simplaway PRODUCTS CO.
(312-359-7337)

P.O. BOX 601, Hoffman Estates, IL 60195
add \$3.00 S&H, 3% for Visa or Mastercard
Illinois Res. Add 6% Sales Tax

WORDSTAR is a trademark of MicroPro INTERN'L CORP.
dBASE is a trademark of ASHTON-TATE CORP.

Circle no. 56 on reader service card.

WHY FORTH ?

- Genuinely Interactive (BASIC is much less interactive)
- Encourages Modular Programs (inefficiency and cluttered syntax hamper effective modularization in compiled languages)
- Fast Execution (not even C is faster)
- Amazingly Compact Code
- Fast Program Development
- Easy Peripherals Interfacing

HS / FORTH

- Fully Optimized for IBM-PC IBM-XT IBM-JR and all PC DOS compatibles
- Forth-79 and Forth-83 Modes
- Full Support for DOS Files, Standard Screens and Random Access DOS Screen Files
- Full Use of 8088 Instructions (not limited 8080 conversion subset of transported versions)
- Separate Segments for Code, Stack, Vocabularies, and Definition Lists - multiple sets possible
- Segment Management Support
- Data Anywhere in Full Megabyte
- Coprocessor Support
- Multi-task, Multi-user Compatible
- Automatic Optimizer (no assembler knowledge needed)
- Full Assembler (interactive, easy to use & learn)
- Goal Oriented Documentation
- Free Updates and News until June 1985
- BYTE Sieve Benchmark jan83
HS/FORTH 47 sec BASIC 2000 sec
w/AUTOOPT 9 sec Assembler 5 sec
other Forths (mostly 64k) 70-140 sec
PS You don't have to understand this ad to love programming in HS/FORTH!

HS/FORTH with AUTO-OPT &
MICRO-ASM \$220.

HARVARD SOFTWARES

PO BOX 339
HARVARD, MA 01451
(617) 456-3021

PL/C Compiler Listing

Listing Two (Listing continued, text begins on page 44)

```
; scan item
X0013:  XFER      X0015,4      ; scan call
        XFER      X0016,4      ; macro call
        XFER      X0017,4      ; scan string
        XFER      X0018,4      ; scan value
        XFER      X0019,0      ; parenthetic group
        NODE      1
        RTRN

; scan call
X0015:  XFER      X0009,0      ; compiler label
        XFER      X001A,1      ; modifier
        SYMB      2
        STOP      `      XFER  `
        NODE      1
        STOP      `.`
        NODE      2
        STOP      `
        NODE      3
        STOP      `

        RTRN

; macro call
X0016:  SKIP
        SCAN      `&`.0
        XFER      X001B,4      ; built in
        XFER      X001C,0      ; user macro
        NODE      1
        RTRN

; built in
X001B:  XFER      X001D,4      ; next
        XFER      X001E,4      ; member
        XFER      X001F,0      ; collect
        NODE      1
        RTRN

; next
X001D:  SCAN      `NEXT`,4
        SCAN      `next`.0
        SKIP
        SCAN      `(`.0
        XFER      X0020.0      ; string
        SKIP
        SCAN      `)` .0
        XFER      X001A,1      ; modifier
        STOP      `      NEXT
        NODE      3
        STOP      `.`
```

(Continued on page 60)

SWIG[©]

SOFTWARE WRITERS INTERNATIONAL GUILD

THE LARGEST PAID MEMBERSHIP PROGRAMMERS GUILD -
OVER 5,000 MEMBERS WORLDWIDE!!

SCHEDULED SWIG ACTIVITIES & MEMBERSHIP BENEFITS

- (1) \$10,000 PROGRAMMING CONTEST (Members only)
- (2) NATIONAL COMPUTER WEEK (May 11 - May 20, 1984)
- (3) ANNUAL CONFERENCE AND SOFTWARE AWARDS CEREMONY (During National Computer Week)
- (4) CONSULTANT REGISTRY (With computer store referral system for customized software)
- (5) JOB PLACEMENT SERVICE (Free to individual members, fixed maximum fee to companies)
- (6) FREE SEMINARS & MEETINGS LOCALLY
- (7) SOFTWARE LIBRARY LENDING & EXCHANGE SERVICE (Professional quality assemblers, utilities, games, etc.)
- (8) SOFTWARE LOCATION SERVICE (For companies & individuals-if it exists, **SWIG** will find it. If not, see #9)
- (9) SOFTWARE DEVELOPMENT SERVICE (From novice to scientist, **SWIG** members can work on any project-from applications to games to R&D)
- (10) LEGAL SERVICE
- (11) AGENT (**SWIG** can represent you in sales to software publishers)
- (12) 24 HOUR - 7 DAY BULLETIN BOARD SYSTEM (BBS) ACCESSIBLE BY COMPUTER FREE
- (13) AND MORE!!!!

MEMBERSHIP APPLICATION FOR SOFTWARE WRITERS INTERNATIONAL GUILD

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

PHONE # () _____

• CLASSIFICATION:

- ☐ NOVICE ☐ BEGINNER TO ADVANCED
- ☐ ADVANCED WITH ON THE JOB EXPERIENCE ☐ RESEARCH/SCIENTIST

• WHAT EQUIPMENT DO YOU HAVE EXPERIENCE WITH &/OR ACCESS TO &/OR PLAN TO BUY?

- ☐ MAINFRAME ☐ MINI ☐ MICRO ☐ DESIGN/R&D
- BRAND NAME(S): ☐ IBM ☐ XEROX ☐ APPLE ☐ TI
- ☐ COMMODORE ☐ RADIO SHACK ☐ ATARI ☐ OSBORNE
- ☐ TIMEX/SINCLAIR ☐ NORTH STAR ☐ HEWLETT PACKARD
- ☐ OTHER _____

• AREAS OF INTEREST:

- ☐ DATA PROCESSING ☐ BUSINESS APPLICATIONS ☐ GRAPHICS
- ☐ LEGAL ☐ VOICE ☐ MEDICAL ☐ APPLIANCE (HOME) CONTROL
- ☐ ROBOTICS ☐ GAMES ☐ MUSIC ☐ R&D ☐ OTHER _____

• MEMBERSHIP ACTIVITIES AND SERVICES OF INTEREST:

READ THE LIST ON THE LEFT AND CIRCLE THE NUMBERS BELOW THAT APPLY.

1 2 3 4 5 6 7 8 9 10 11 12

- ☐ I HAVE ENCLOSED \$20 ANNUAL MEMBERSHIP FEE ☐ CK ☐ MO
(MAKE CHECK PAYABLE TO: **SWIG**)

RETURN TO: **SWIG**
P.O. BOX 87
STONY POINT, NEW YORK 10980
(914) 354-5585

SWIG © SOFTWARE WRITERS INTERNATIONAL GUILD

MicroMotion

FORTH

The Professional Choice.
Make it yours.

Everything for the FORTH user
from start to finish.

- Meets the 1983 International standard for portability
- 200 page tutorial and reference manual
- Available for APPLE II and Z-80 CP/M (many disc formats)
- Floating point extension
- HIRES graphics extension (APPLE II & North Star Advantage only)
- Screen editor with definable controls
- Macro assembler with local labels
- Applications
- MicroMotion FORTH - \$90. - \$140.
- Publications
 - FORTH-83 International Standard
 - FORTH-83 compatible source listings for 6502, 8080, Z-80 and 8086 microprocessors
 - FORTH-83 Standard - \$15.00
 - Source listings - \$20.00

Contact us for our most up-to-date
product information!



MicroMotion
12077 Wilshire Blvd., Ste. 506
Los Angeles, CA 90025
(213) 821-4340

PL/C Compiler Listing

Listing Two

(Listing continued, text begins on page 44)

```

      NODE      1
      STOP      ,

      RTRN

; member

X001E:  SCAN      `MEMB`,4
        SCAN      `memb`,.0
        SKIP
        SCAN      `(`.0
        XFER      X0020.0          ; string
        SKIP
        SCAN      `)`0
        XFER      X001A,1          ; modifier
        STOP      , MEMB
        NODE      3
        STOP      ,
        NODE      1
        STOP      ,

      RTRN

; collect

X001F:  SCAN      `COLL`,.4
        SCAN      `coll`,.0
        SKIP
        SCAN      `(`.0
        XFER      X0021,4          ; col on
        XFER      X0022.0          ; col off
        SKIP
        SCAN      `)`0
        STOP      , COLL
        NODE      2
        STOP      ,

      RTRN

; col on

X0021:  SCAN      `ON`,.4
        SCAN      `on`,.0
        STOP      `ON`
        RTRN

; col off

X0022:  SCAN      `OFF`,.4
        SCAN      `off`,.0
        STOP      `OFF`
        RTRN

```

(Continued on page 62)

LIFEBOAT™ Associates:
The full support software
source.

Reach for the programming horizon of the 80's with Lattice C, the fastest C compiler.

Set the course of your next software project towards greater power and portability by selecting Lattice C, recognized as the finest and fastest 16-bit C compiler. Lattice C, the full implementation of Kernighan and Ritchie, continues to lead the way by announcing full memory management up to 1 Megabyte. Major software houses including Microsoft, MicroPro™ and Sorcim™ are using Lattice C to develop their programs.

Lattice C is available for a wide variety of 16-bit personal computers including IBM®, NCR®, Texas Instruments, Victor, Wang and other microcomputers running PC™ -DOS, MS™ -DOS and CP/M86™.

Call LIFEBOAT at 212-860-0300 for free information on the Lattice C family of software development tools.

LIFEBOAT

Associates

Lifeboat offers Lattice C with a tested set of software tools to provide a complete development system including:

HALO™
PANEL™
PMATE™
PLINK™-86
C-FOOD SMORGASBORD™
LATTICE WINDOW™
FLOAT87™

Color graphic primitives
Screen design aid
Customizable program editor
Overlay linkage editor
Screen and I/O utilities
Multi-window functions
8087 floating point math

LIFEBOAT

Associates

1651 Third Avenue
New York, NY 10028
212-860-0300

Please send me free information on: Name

- ☐ Lattice and development tools
- ☐ Corporate purchase program
- ☐ Dealer program
- ☐ OEM agreements

- ☐ Send me the complete LIFEBOAT software catalog. \$3.00 enclosed for postage and handling.

Company

Address

City

State

Zip

Telephone

LATTICE, C-FOOD SMORGASBORD and
LATTICE WINDOW, TM.; Lattice, Inc.
MICROPRO, TM International Corp.
SORCIM, TM Sorcim Corp.

LIFEBOAT, TM Intersoft Corp.
HALO, TM Media Cybernetics
PANEL, TM Roundhill Computer, Ltd
PMATE and PLINK, TM Phoenix Software

FLOAT87, TM Microfloat
IBM and PC, *TM International Business Machines
MS, TM Microsoft
CP/M86, TM Digital Research

PL/C Compiler Listing (Listing continued, text begins on page 44)

Listing Two

```
; user macro

X001C:  XFER      X000A,0          ; label root
        XFER      X0023,1          ; macro parameter list
        STOP
        NODE      2
        NODE      1
        STOP
        RTRN

; macro parameter list

X0023:  SKIP
        SCAN      `(`.0
        XFER      X0024,0          ; macro arg
        XFER      X0025.3          ; addl macro arg
        SKIP
        SCAN      `)`'.0
        STOP
        NODE      3
        NODE      2
        RTRN

; macro arg

X0024:  XFER      X0020.4          ; string
        XFER      X0026,4          ; macro value
        XFER      X000A,0          ; label root
        NODE      1
        RTRN

; string

X0020:  SKIP
        SCAN      ``'.0
        COLL      ON
        XFER      X0027,2          ; string character
        COLL      OFF
        SCAN      ``'.0
        OHEX      '60'
        NODE      2
        OHEX      '60'
        RTRN

; string character

X0027:  XFER      X000D,4          ; alphanumeric
        XFER      X0028,4          ; special
        XFER      X0029,0          ; string quote
        NODE      1
        RTRN
```


; special

X0028: MEMB ` !"#\$\$%&()*+:=-{[]}`~^+;@|\ <,>./?
`.0
NODE 1
RTRN

; string quote

X0029: COLL OFF
SCAN `'''`.0
COLL ON
STOP `'
RTRN

; macro value

X0026: SKIP
COLL ON
XFER X0010.2 ; digit
COLL OFF
NODE 1
RTRN

; addl macro arg

(Continued on next page)

C BUILDING BLOCKS

Save a year of development.

- ☐ **IBM PC* Building Blocks** \$150
Video & IO routines, string functions, asynchronous port control, date, time, random numbers, printer control. Over 200 functions.
- ☐ **Mathematics Library** \$ 66
Logarithms, trigonometric functions, square root, random numbers.
- ☐ **C Cross-reference Utility** \$ 99
Cross-reference identifiers and functions.
- ☐ **Communications Library** \$150
For Hayes Smartmodem†, modem 7 and xmodem.
- ☐ **B-trees & Virtual Memory Management Library** ... \$125
List handling features.
- ☐ **Advanced Building Blocks** \$ 99
Julian dates, dBaseII access, data compression, text windows.

All Building Blocks use the 'IBM PC Building Blocks'. Source Included. Credit Cards Accepted. Single User License. Multiuser licenses available.

具 NOVUM
ORGANUM

29 Egerton Road, Arlington, MA 02174 Tel: (617) 641-1650
(TM) IBM* (TM) Hayes Microcomputer Products† (TM) Ashton-Tate*

Circle no. 39 on reader service card.



the best VALUE anywhere COMMUNICATIONS SOFTWARE PACKAGE COMM-X-PAC™

- COMM-X Smart Terminal and File Transfer Auto Dial and Logon Files for Easy Linkup Protocols for Timeshare, MODEM 7, CRC16 FORTRAN version available for Mainframes
- Unattended "Midnite" Electronic Mail Subsystem
- CONSOLX Remote Operation of Your Computer
- Bulletin Board System with Database Access
- Database File Management System Included
- Utilities for Data Encryption, Compression, and More
- CPU License \$150 Detailed Manual \$25



HAWKEYE GRAFIX

Contact Your Local Dealer or
Call or Write For Free Brochure

23914 Mobile, Canoga Park, Ca. 91307 • 213/348-7909 U.S.A.

Circle no. 26 on reader service card.

Elegance

Power

Speed



C Users' Group
Supporting All C Users
Box 287
Yates Center, KS 66783

Circle no. 15 on reader service card.

A Professional Quality Z80/8080 Disassembler

REVAS Version 3

Uses either ZILOG or 8080 mnemonics
Includes UNDOCUMENTED Z80 opcodes
Handles both BYTE (DB) & WORD (DW) data
Disassembles object code up to 64k long!
Lets you insert COMMENTS in the disassembly!

A powerful command set gives you:

INTERACTIVE disassembly
Command Strings & Macros
On-line Help
Calculations in ANY Number Base!
Flexible file and I/O control
All the functions of REVAS V2.5

REVAS:

Is fully supported with low cost user updates
Runs in a Z80 CPU under CP/M*
Is normally supplied on SSSD 8" diskette
Revass V 3...\$90.00 Manual only...\$15.00
California Residents add 6½% sales tax

REVASCO

6032 Chariton Ave., Los Angeles, CA. 90056
(213) 649-3575

*CP/M is a Trademark of Digital Research, Inc.

Circle no. 51 on reader service card.

PL/C Compiler Listing

Listing Two

(Listing continued, text begins on page 44)

```
X0025:  SKIP
        SCAN      \'.0
        XFER      X0024.0      ; macro arg
        STOP      \'.
        NODE      1
        RTRN
```

; scan string

```
X0017:  XFER      X0020.0      ; string
        XFER      X001A,1      ; modifier
        STOP      \'.      SCAN
        NODE      2
        STOP      \'.
        NODE      1
        STOP      \'
```

RTRN

; scan value

```
X0018:  SKIP
        SCAN      \#'.0
        SCAN      \'.0
        XFER      X002A,2      ; hex pair
        SKIP
        SCAN      \'.0
        XFER      X001A,1      ; modifier
        STOP      \'.      SCNR
        OHEX      '60'
        NODE      3
        OHEX      '60'
        STOP      \'.
        NODE      1
        STOP      \'
```

RTRN

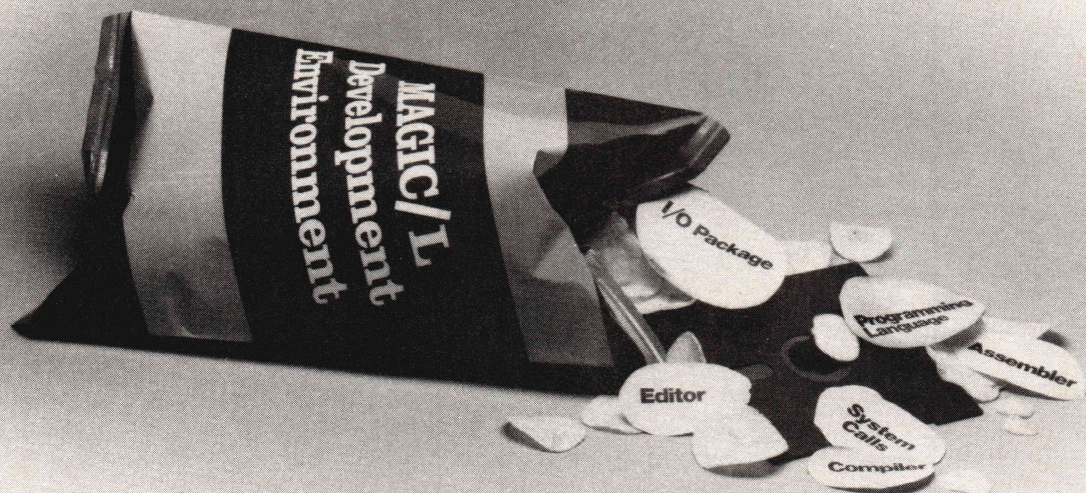
; hex pair

```
X002A:  SKIP
        COLL      ON
        XFER      X002B,0      ; hex digit
        XFER      X002B,0      ; hex digit
        COLL      OFF
        NODE      2
        NODE      1
        RTRN
```

; hex digit

(Continued on page 66)

USE MAGIC/LTM TO WRITE JUST ONE PROGRAM



Bet you can't stop with one.

For just \$295 MAGIC/LTM is yours. And suddenly you find you've bought more than a language. MAGIC/L is an entire interactive environment; an assembler, a compiler, an editor, an I/O package, and a system call facility are all wrapped up and delivered. And now MAGIC/L is available for CP/M-based computers.

MAGIC/L is easy to learn. It has syntax similar to C and Pascal, and because it's extensible as well as interactive, it dramatically increases productivity.

Program development features include a built-in text editor, command line recall, CCP, STAT, and PIP command emulation, and the ability to store keyboard dialog on disks.

Key language features include: CHAR, INTEGER, LONG, REAL, and String data types; record structures similar to the STRUCT facility in C; and a complete I/O package that can provide random access, variable length I/O to any CP/M file.

And MAGIC/L offers great portability. Source code which runs on CP/M can be compiled unmodified and run on any other processor.

Typical applications include hardware interfacing, process control, games creation, interactive graphics and image processing. MAGIC/L has made programming easier for DEC, 68000, and Data General users. Now it's working for CP/M users too.

MAGIC/L provides everything you need to write a complete program. But the only way to be convinced is to try it yourself. Send us your \$295 check or money order—we also accept MasterCard and VISA—we'll send MAGIC/L for CP/M to you at once. A full money back guarantee is part of the package. Once you've sampled that first program, you'll have to try another . . . and another . . . and another. MAGIC/L . . . it's more than a language.

MAGIC/L . . . It's more than a language



LOKI ENGINEERING, INC.

55 Wheeler St., Cambridge, MA 02138 (617) 576-0666

Circle no. 30 on reader service card.

MAGIC/L is a trademark of Loki Engineering, Inc.
DEC is a trademark of Digital Equipment Corporation.
CP/M is a trademark of Digital Research, Inc.

PL/C Compiler Listing

(Listing continued, text begins on page 44)

Listing Two

```
X002B: MEMB    `0123456789ABCDEF`.0
      NODE    1
      RTRN

; parenthetic group

X0019: SKIP
      SCAN    `(`.0
      XFER    X0011,2      ; action specification
      SKIP
      SCAN    `)`0
      XFER    X001A,1      ; modifier
      LGEN
      LGEN
      STOP    `      XFER `
      NODE    2
      STOP    `:`
      NODE    3
      STOP    `
      GOTO    `
      NODE    1
      STOP    `

      NODE    2
      STOP    `:`
      NODE    5
      STOP    `      RTRN

      NODE    1
      STOP    `:`
      RTRN

; parenthetic body

X002C: XFER    X0011,0      ; action specification
      SKIP
      SCAN    `)`4
      XFER    X002C,0      ; parenthetic body
      NODE    3
      NODE    1
      RTRN

; modifier

X001A: SKIP
      XFER    X002D,4      ; alt mod
      XFER    X002E,4      ; optionally many
      XFER    X002F,4      ; one or more
      XFER    X0030,4      ; optional
      XFER    X0031,0      ; default zero
      NODE    1
      RTRN
```


; alt mod

```
X002D:  XFER      X0032.4      ; multi alt
        XFER      X0033,0      ; single alt
        NODE      1
        RTRN
```

; multi alt

```
X0032:  SCAN      `$.0
        SKIP
        SCAN      `1`.0
        STOP      `6`
        RTRN
```

; single alt

```
X0033:  SCAN      `1`.0
        STOP      `4`
        RTRN
```

; optionally many

```
X002E:  SCAN      `$.0
        SKIP
        SCAN      `*.0
        STOP      `3`
        RTRN
```

; one or more

```
X002F:  SCAN      `$.0
        STOP      `2`
        RTRN
```

; optional

```
X0030:  SCAN      `*.0
        STOP      `1`
        RTRN
```

; default zero

```
X0031:  STOP      `0`
        RTRN
```

; output item

```
X0014:  SKIP
        SCAN      `[`.0
        XFER      X0034.2      ; output spec
        SKIP
        SCAN      `]`.0
        NODE      2
        RTRN
```

; output spec

(Continued on next page)

uniforth

One of the finest implementations of the FORTH language. Field tested and reliable, **UNIFORTH** is available for the DEC Rainbow/Professional, Osborne, KayPro, and IBM PC as well as most systems with 8" disks and the following processors:

8080	PDP-11
Z80	68000
8086/8	16032

As a task, **UNIFORTH** is compatible with and supports all features and file types of the CP/M, CDOS, MS-DOS and DEC operating systems. As an operating system, **UNIFORTH** will function "stand-alone" on most commercial microcomputers.

The FORTH-79 Standard language has been extended with over 500 new words that provide full-screen and line-oriented editors, array and string handling, enhanced disk and terminal I/O, and an excellent assembler. Detailed reference manuals supply complete documentation for programming and system operation, in an easy-to-understand, conversational style using numerous examples.

Optional features include an excellent floating-point package with all transcendental functions (logs, tangents, etc.), the MetaFORTH cross-compiler, printer plotting and CP/M file transfer utilities, astronomical and amateur radio applications, word processing, etcetera.

Compare these features with any other FORTH on the market:

- Speed and efficiency
- Ease of use
- Variety of options
- Documentation quality

You'll find **UNIFORTH** is superior.

Prices start at \$35. Call or write for our free brochure.

Unified Software Systems

P.O. Box 2644, New Carrollton, MD 20784, (301) 552-9590

CP/M Digital Research, CDOS Cromenco, DEC PDP Digital Equipment Corporation, MSDOS Microsoft, IBM PC IBM, Z80 Zilog

Circle no. 65 on reader service card.

GGM — FORTH™ has HELP* for Z80¹ using CP/M²

GGM—FORTH, a complete software system for real-time measurement and control, runs on any Z80 computer under CP/M using an extended fig-FORTH vocabulary.

GGM—FORTH features:

- Open multiple CP/M files, in any combination of direct-access and sequential-access, fully compatible with all CP/M utilities
- Char. in/out uses CP/M console, lister, file, or port
- On-line HELP* provides instant access to definitions in the run-time GGM—FORTH dictionary
- HELP* file is easily extended to include user definitions using HELP* utility
- HELP* is available during full-screen editing

Complete system and manuals	\$150.
Manuals only:	\$ 20.
Introductory System:	\$ 35.

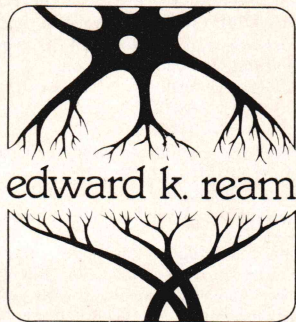
GGM SYSTEMS, INC. (617) 662-0550
135 Summer Ave., Reading, MA 01867

¹ Z80 is a trademark of Zilog, Inc.

² CP/M is a trademark of Digital Research, Inc.

Circle no. 23 on reader service card.

RED



FULL SCREEN EDITOR with FULL SOURCE CODE in C for CP/M 68K or CP/M 80

- RED is a powerful yet simple text editor for both programmers and writers.
- RED features 17 commands including block move, block copy, search and substitute.
- RED is the only text editor supplied with full source code for the BDS C, Aztec CII and Digital Research C compilers.
- RED is the only text editor that you can change to suit your needs and tastes.
- RED is the only text editor that will not become obsolete when you change systems.
- RED supports all features of your terminal. You tailor RED to your terminal with an easy-to-use configuration program.
- RED handles files as large as your disk.
- RED is guaranteed. If for any reason you are not satisfied with RED, your money will be refunded promptly.

Price: \$50.

For more valuable information, call (608) 231-2952

To order, send \$50 to:
Edward K. Ream
1850 Summit Avenue
Madison, Wisconsin 53705

Your order will be mailed to you within one week. Sorry, I can not handle credit cards. Please do not send purchase orders unless a check is included.

Dealer inquiries invited.

PL/C Compiler Listing Listing Two

(Listing continued, text begins on page 44)

```
X0034:  XFER      X0035.4      ; node spec
        XFER      X0036,4      ; output string
        XFER      X0037,0      ; output value
        NODE      1
        RTRN
```

; node spec

```
X0035:  XFER      X0038,0      ; node number
        XFER      X0039,3      ; addl node number
        NODE      2
        NODE      1
        RTRN
```

; node number

```
X0038:  SKIP
        COLL      ON
        XFER      X0010.2      ; digit
        COLL      OFF
        STOP      '          NODE '
        NODE      1
        STOP      '
        RTRN
```

; addl node number

```
X0039:  SKIP
        SCAN      '.\'.0
        XFER      X0038,0      ; node number
        NODE      1
        RTRN
```

; output string

```
X0036:  XFER      X0020.0      ; string
        STOP      '          STOP '
        NODE      1
        STOP      '
        RTRN
```

; output value


```

X0037:  SKIP
        SCAN      `<`.0
        SKIP
        XFER      X002A,2      ; hex pair
        SKIP
        SCAN      `>`.0
        STOP      `      OHEX      `
        NODE      2
        STOP      `

```

RTRN

; semi

```

X0012:  SKIP
        SCAN      `;`.0
        RTRN

```

; end

```

X0003:  SKIP
        SCAN      `END`,4
        SCAN      `end`,0
        XFER      X0012.0      ; semi
        STOP      `      .END

```

```

OHEX      '1A'
RTRN
.END

```

End Listing



"Today, class, I would like to explain the advantages of the hexadecimal system."

You Read Dr. Dobb's Journal And You Don't Subscribe?!

**Save over \$23.00
off newsstand prices for 2 yrs.
Save over \$10.00 for 1 yr.**

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

DR. DOBB'S JOURNAL
P.O. Box E
Menlo Park, CA 94026

Yes! Sign me up for
 ___ 2 yrs. \$47 ___ 1 yr. \$25
 ___ I enclose a check/money order
 ___ Charge my Visa, MasterCard,
 American Express
 ___ Please bill me later

Name _____

Address _____

_____ Zip _____

Credit Card _____ Exp. date _____

Account No. _____

Signature _____

N5

Circle no. 71 on reader service card.

Program Design Using Pseudocode

Programming involves two distinct steps: design and coding. Programmers are often quite adept at coding but deplorably inadequate when it comes to working out the design part.

When you code, your main concern is the language and all the details of program construction: how to pass parameters to the function, how to set up the search loop, or whether the loop condition is checked at its beginning or end. It's something like taking a trip. Which road should we take? When should we stop for gas? What if the Disneyland Hotel has no vacancies?

Designing is analogous to planning the trip. Where shall we go? What shall we do? Shall we drive or fly? Unfortunately, planning the trip, like wrestling with the design of a program, is hardly ever as exciting as actually *doing* it. It must be human nature to say, "Aw, heck, I'll figure it out as I go." The result is usually pretty chaotic. "Let's see, was INREC 8 bytes or 10 bytes long? I assumed 8 here, but I used 12 over there. . . ."

Except for a really trivial program, it is unreasonable to assume that you can design while coding or code while designing. Usually, the design loses out since you tend to concentrate more on coding detail (the *how* part) than on design detail (the *what it does* part).

Numerous tools are available for designing programs. Flow charts are probably the most heavily abused of them. Some structured design methodologies, complete with a plethora of rigid rules for their usage, are also available. Then there is pseudo-code.

Pseudo-code is a sort of "fake" programming language, one that is meant to give you lots of flexibility in designing a program. At its worst, pseudo-code can be so restrictive that it becomes much like another programming language. At its best, it allows you to create structures that make sense only to you, although they may be meaningless to a compiler.

In this article, I will present a way of using pseudo-code during the design process. Although I don't intend to set down any coding conventions, I will describe the conventions I used at the end

"Pseudocode is a sort of 'fake' programming language, one that is meant to give you lots of flexibility in designing a program."

of the article. Our concern here is the method of usage, not the method of coding.

Design Philosophy

We are going to design a game program. We will start by describing it in very general, rather vague terms. We will then build a modular design using structured English, working from the top down. We will ignore such things as subroutines, variables (for the most part), structures, and primitives. It won't look anything like a program.

Then we will get into pseudo-code. We will translate each module into a routine using pseudo-code, creating more routines as we go. We will also start creating variables, both local and global, and we will reference several data structures. Because of the inexact nature of the design process, we will probably reiterate the various stages several times, reorganizing modules and rebuilding data structures. After that, we'll think about coding.

A Shoot-'Em-Up Game

You are playing a video game. In the center of the screen is the cross hair, against a background of fixed stars. As you move the joystick around, the background moves also. It's as if you were in a spaceship, in the gunner's compartment, scanning nearby space with your missile launcher. A series of fighter craft comes into view! They weave about against the fixed background, and the whole scene moves about the screen as you maneuver the joystick to take aim. When you press the fire button, a missile is launched. Each time a missile intercepts an attacking fighter, you see an explosion and are awarded points. You play until a certain number of ships gets past you, at which point the game ends. The number of fighters on the screen during a "wave" may be limited, as well as the number of missiles on the screen at a time.

Does it sound exciting? No? Not a video game addict, eh? Well, bear with me. The principles we will use to design

this game are applicable elsewhere, too. Let's now do what I call a Level 1 design specification, using structured English. We may have to rework the design several times, but we should have a decent one by the time we're done.

Level 1 Design

Designing usually requires lots of paper, most of which ends up in the trash. We'll skip the scribbling and go straight to the final draft. Our top down design begins with the main module.

Main Module

1. Create attacking fighters and set their initial positions.
2. Set initial coordinates of cross hairs against the reference background.
3. Draw the initial screen.
4. Until the number of fighters that "escape" is greater than the limit, do the following:
 - a. See if any missiles are fired (Fire Control).
 - b. Check for interceptions (Intercept Control).
 - c. Move the fighters (Fighter Movement).
 - d. Move any missiles (Missile Control).
 - e. Determine cross hair position (Scan Control).
 - f. Draw new screen (Display Module).
 - g. Delay for skill level.
 - h. Loop back to 4.
5. End of game; display final tally.

Fighter Movement

Given that we know the previous x,y position of each fighter and its range from us:

For each fighter, calculate a new x,y position against the reference background and a new range.

Missile Control

Given that we know the previous

by Ken Takara

Ken Takara, 1306 Sippola Way, San Jose, CA 95121.

range of the missile from the player (and assuming that its x,y reference coordinates don't change):

Calculate its new range.

Scan Control

Given that we know the previous x,y position of the cross hairs against the reference background:

1. Read the x,y values of the joystick and convert to $(\pm 1, \pm 1)$ values.
2. Change the x,y values by adding the results from the joystick. This implies that moving the joystick causes the center of the screen to drift against or "scan" the reference background.

Intercept Control

1. For each missile:
 - a. Get its x,y position and range.
 - b. Calculate the range it will have after its next missile movement, and call it range 1.
2. For each fighter whose x,y position is the same as that of the missile, if missile range \geq fighter range \geq range 1:
 - a. Get the point value of the fighter.
 - b. Eliminate the missile and fighter.
 - c. Mark an explosion at the fighter's location.
3. Return the total score.

We'll spare you the rest of the modules; suffice to say that Fire Control checks the fire button, and Display Module displays the screen, showing fighters, missiles, explosions, and background stars.

There are quite a few implications and assumptions here. We imply that a data structure exists for each of the fighters, missiles, and explosions. We imply variables for keeping the score and constants to regulate the game speed and number of rounds to play. We assume that we can read the joystick and convert its values and that we can read the fire button.

It's a good idea to sit back and reconsider the game in its entirety just so that we don't become too myopic from staring at the details. For example, notice that Intercept Control checks a missile's next position. The Missile Control module also needs this information in order to move the missile. We might want to calculate the position once then save it for use by other modules. We should make a note of this and modify the design to take this into account. If we noticed this while coding, major surgery might be required on our beautifully written program to accommodate this discovery.

Notice also that, despite all these assumptions and implications, we have not specified how any of this will be done. We simply assume that it can be done. As programmers, we ought to know if it

can be done or not, and we ought to know the limits of the machine. In any event, we have specified what it does without worrying about how it will do it.

At this point, we can easily change our minds concerning the design of the game. We can let the fighters shoot back, or we can describe various classes of fighters with different abilities. We have this flexibility because we have yet to set up the rigid data and control structures.

Once we are satisfied with our base idea, we go on to something closer to a program. We can start using data structures and variables and flow-of-control,

and leave the somewhat ambiguous structured English of Level 1. Figure 4(a) (page 75) presents the conventions adopted for Level 1 design for use as guidelines in subsequent design efforts.

Level 2 Design

During Level 1, we described a set of modules in rather loose terms, implying all sorts of data and control structures, as well as numerous functions and sub-routines, without really acknowledging them as such. In Level 2, we transform the modules into routines using real,

PRICE BREAKTHROUGH

The wait-loss experts have done it again!

512Kbyte SemiDisk™ with SemiSpool™

\$1095

Time was, you thought you couldn't afford a SemiDisk. Now, you can't afford to be without one.

	256K	512K	1Mbyte
SemiDisk I, S-100	\$895	\$1095	\$1795
IBM PC		\$1095	\$1795
TRS-80 Mdl. II, CP/M		\$1095	\$1795
SemiDisk II, S-100		\$1395	\$2095
Battery Backup Unit	\$150		
Version 5 Software Update	\$30		

Time was, you had to wait for your disk drives. The SemiDisk changed all that, giving you large, extremely fast disk emulators specifically designed for your computer. Much faster than floppies or hard disks, SemiDisk squeezes the last drop of performance out of your computer.

Time was, you had to wait while your data was printing. That's changed, too. Now, the SemiSpool print buffer in

our Version 5 software, for CP/M 2.2, frees your computer for other tasks while data is printing. With a capacity up to the size of the SemiDisk itself, you could implement an 8 Mbyte spooler!

Time was, disk emulators were afraid of the dark. When your computer was turned off, or a power outage occurred, your valuable data was lost. But SemiDisk changed all that. Now, the Battery Backup Unit takes the worry out of blackouts.

But one thing hasn't changed. That's our commitment to supply the fastest, highest density, easiest to use, most compatible, and most cost-effective disk emulators in the world.

SemiDisk.

It's the disk the others are ^{still} trying to copy.

SEMIDISK SYSTEMS, INC.

P.O. Box GG Beaverton, OR 97075 (503) 642-3100

Call 503-646-5510 for CBBS®/NW, a SemiDisk-equipped computer bulletin board. 300/1200 baud
SemiDisk, SemiSpool Trademarks of SemiDisk Systems. CP/M Trademark Digital Research



Circle no. 54 on reader service card.

live pseudo-code. Although we use variables, data structures, and certain control structures, we aren't really coding in the normal sense. We are still describing what the routines do but in sufficient detail to make the how of it easy, even obvious, to work out.

We'll start by identifying what might be global variables. These are data structures and other variables that seem to be used in different modules, retaining the changes that other modules make to them. For example, the fighters might be described by a collection of data structures called SHIPREC, while missiles are described by a collection of data structures called MISSILE. Some obvious variables are the total number of points, current center-of-screen coordinates, and the number of fighters that have "escaped."

For Level 2 designs for the Main Module and the Intercept Control routines, we'll again skip the scratchpaper stage and present a (nearly) finished draft. Figure 1 (page 72) shows the logic for the Main Module.

The Main Module routine uses a number of routines, most of which are familiar from Level 1. The notation SHIPREC (i) does not imply that SHIPREC is an array; it is simply a convenient fiction to indicate "this particular SHIPREC" as opposed to "any SHIPREC." We have described only those variables and constants actually used by this routine, although many other global variables exist; we will indicate global variables in other routines as we come to them to keep our list of variables and constants from getting out of hand. We didn't need to describe each of the routines derived from the modules, but we did it for completeness.

We also used two loop structures, one of which looks vaguely like the "do until" loop. The other is easy to understand but looks unlike any "real" loop structure in any language. The two loop structures are actually pseudo-loops, in that they aren't supposed to tell you how to code the loop or even which loop to use. They only tell you what to loop on and when to quit the loop. More on this later.

Of course, the top level module of many programs can be awfully dull; nothing really happens there. All the action is found in the lower level routines. Figure 2 (page 74) shows the pseudo-code for the Intercept Control routine.

The MX, MY, MR, and MR1 variables are local and temporary. Depending upon the language, they may be represented by registers, values on stacks, etc. In some cases, they may never be really created.

The functions, XPOSITION.M, YPOSITION.M, XPOSITION.S, and so on may be fictitious, depending upon the actual format of the fighter records and missile records. They are created here

Main Module Logic

```

MAIN MODULE: routine;
  for each SHIPREC (i) do                                (init fighter records)
    call NEWSHIP (SHIPREC (i));                          (create ship record)
  loop (i);                                                (next rec)
  FAILURE.COUNT := 0;                                     (failure counter)
  call DISPLAY.SCREEN;                                    (draw 1st screen)
  until FAILURE.COUNT=MAXFAIL do                          (main prog loop)
    call FIRE.CONTROL;                                    (launch missiles)
    SCORE := SCORE + INTERCEPT;                        (Intercept Ctl)
    call FIGHTER.MOVEMENT;                                (move fighters)
    call MISSILE.CONTROL;                                 (move missiles)
    call SCAN.CONTROL;                                    (new x,y cross hairs)
    call DISPLAY.SCREEN;                                  (draw new screen)
    call SPEED.DELAY;                                    (slow game)
  again;
  call END.GAME;                                          (final display)
end MAIN MODULE;

```

Variables and Constants

SHIPREC

Global structure giving positions of fighters (x,y, range) as well as other information.

SCORE

Global variable with current score.

FAILURE.COUNT

Variable giving number of ships that have "escaped."

MAXFAIL

Constant giving maximum number of failures before game ends. Suggested value=10.

Functions and Routines

NEWSHIP (ship record)

Initializes a record, creating a new fighter, position, value, etc.

DISPLAY.SCREEN

Draws a screen with current objects (fighters, missiles, explosions, stars, etc.).

FIRE.CONTROL

Performs missile launches.

INTERCEPT

Returns total point value of any fighters shot down by missiles. Checks for interception and eliminates hit fighters and missiles. Records explosions.

FIGHTER.MOVEMENT

Moves each of the fighters.

MISSILE.CONTROL

Moves each of the missiles.

SCAN.CONTROL

Checks joystick and sets new center-of-screen coordinates for cross hairs.

SPEED.DELAY

Slows down the action by inserting a delay.

END.GAME

Performs end-of-game cleanup and display.

Figure 1.

Intercept Routine Logic

```

INTERCEPT: Function;
POINTVAL := 0;                                (initialize point count)
for each (MISSILE(i)) do                       (another loop)
    MX := MISSILE(i).XPOS;                     (current x coordinate)
    MY := MISSILE(i).YPOS;                     (current y coordinate)
    MR := MISSILE(i).RANGE;                   (range from player)
    MR1 := MR + MISSILE(i).SPEED               (find next position)
    for each (SHIPREC(j)) do                   (check each target ship)
        if SHIPREC(j).XPOS=MX                 (is fighter at the same . . .)
        and SHIPREC(j).YPOS=MY                (position as missile?)
        then
            if MR<=SHIPREC(j).RANGE<=MR1 then (a hit!)
                POINTVAL := POINTVAL + SHIPREC(j).VALUE
                call ELIMINATE(MISSILE(i));
                call NEWSHIP(SHIPREC(j));
                exitloop(j);                   (exit SHIPREC loop)
            endif
        endif;
    next(j);
next(i);
return (POINTVAL);

```

Variables and Constants

POINTVAL

Sum of points added this turn.

MX, MY, MR, MR1

Holding variables for missile position (x,y,range) and projected range of missile.

SHIPREC

Global data structure describing fighters. Includes position and value information. See description below.

MISSILE

Global structure for missiles. See description below.

MISSILE and SHIPREC Structures

MISSILE(record).XPOS	SHIPREC(record).XPOS
X position of a missile.	X position of an attacking fighter craft.
MISSILE(record).YPOS	SHIPREC(record).YPOS
Y position of a missile.	Y position of a fighter.
MISSILE(record).RANGE	SHIPREC(record).RANGE
Distance of a missile from the player.	Distance from player to attacking fighter.
MISSILE(record).SPEED	SHIPREC(record).VALUE
Velocity at which a missile is traveling.	Point value of a target fighter.
	SHIPREC(record).

Functions and Subroutines

ELIMINATE (missile record)

Deletes a missile record (either by unlinking it, if a linked list is used, or by setting an "unused" flag).

NEWSHIP (ship record)

Replaces a destroyed fighter with a new one.

NEWBANG (x,y,range)

Makes an entry in the explosion table for the display.

Figure 2.

for convenience. The explicit difference between XPOSITION.M and XPOSITION.S, and between YPOSITION.M and YPOSITION.S, and so on, is because we don't know anything about the actual layouts of the missile and fighter records.

The routine NEWSHIP has been re-used, indicating that it may be global or at least defined at a higher level than this routine. When designing and coding it, we will have to keep in mind that it can be called at any time, not only during the initial setup.

Although we have identified some of the fields within the fighter and missile records, we still do not impose a format on them. We may find that we need more fields in some of these records. For these reasons, we reference the fields within the records by name, using the "dot" convention — RECORD.FIELD1 refers to a field called FIELD1 in a data structure called RECORD.

In both of the routines that we pseudo-coded, we referred to the fighter and missile records with miniscule subscripts. This is not intended to imply that they are arrays. We have not specified what their actual structure is and simply use the miniscule subscript to single out a particular record and to imply an ordering to the records. That is, some "first" record exists and for any record a "next" record exists. These records could be arrays, or they could be elements in a linked list or sequential blocks in memory. The physical layout is not important at this time.

Figure 4(b) (page 75) lists the conventions used for Level 2 design.

Loops and Other Control Structures

A dozen different loop constructs no doubt are available among programming languages, as well as a dozen ways to use them with varying degrees of effectiveness. If you use an "until" loop, does it check the condition at the beginning of the loop or at the end? What is the value of the loop counter after the loop terminates? Is the loop always executed at least once?

Many of the loop peculiarities depend upon the implementation. We would like to have a set of pseudo-loops that always does what we want it to do. Since we are working with pseudo-code, there is no reason why we can't create such loops. Let's assume that our loops will be executed zero times if the exit condition is met immediately upon entry to the loop. Let's also assume that loop counters retain their values after the loop is exited. We know that we can ensure these two points during real coding, even if the language doesn't support them, by adding our own checks or by using different loop constructs.

Three things are common to loops.

ACTIVE TRACE

"A marvelous Basic programming aid . . .

It's just amazing to watch a program you wrote run under Scope, and debugging becomes if not trivial, then at least doable."

Thomas Bonoma, *Microcomputing*
Dec. '83, p 22

"Extremely useful program . . . Anyone doing much programming in Basic should appreciate Active Trace a lot."

Jerry Pournelle, *Byte Magazine*
April '83, p 234

Spaghetti code is what many "experts" call a beginner's Basic program which is all tangled up and difficult to follow. The **Active Trace** package will help you learn how to avoid the pitfalls of structureless programs. And if you already have a program which is too confusing to follow, or has an error which is hiding, relax.

Active Trace doesn't get confused. **Active Trace** will lead you through your program letting you know variable values (all variables or just those you specify) as they change. In a form a novice can understand, your program's internal activity is presented on your screen, or printer, or it can be saved on disk.

For more advanced programmers, the disk file of your programs activity can be used with your word processor to automatically find the source of an error and display the circumstances surrounding its occurrence.

Ready to Order?

Just have a Question?

Contact your dealer or call

Toll Free: 800-358-9120(US)

800-862-4948(CA)

Active Trace \$79.95

Complete package includes Scope, XREF mapping and documentation

Scope Separately \$49.95

Only recommended for those who already own professional mapping (XREF) programs

Active Trace is available for most **MS-DOS** and **CPM 2.2** systems and supports the special features of Brand specific versions of Microsoft Basic such as **Basica** on the **IBM-PC**.

AWARECO
Active Software

P.O. Box 695 Gualala, CA 95445
(707) 884-4019

Active trace, Active software, AWARECO and Scope are trademarks of A Ware Company—CPM is a trademark of Digital Research—MS-DOS and Microsoft are trademarks of Microsoft Corporation—IBM-PC is a trademark of IBM Corp.

First, they assume some initial condition or initial item: the initial value of a counter, for example, or the first element in a linked list. Second, most loops assume that a "next" item follows the first. Third, the program can exit the loop either when the elements are exhausted or upon realization of an "exit" condition of some sort. Regardless of the peculiarities of any language, loop structures almost always have provision for these three points. Of course, we may have to do some twisting to use them the way we want to, but that is why we work out the design in pseudo-code to begin with.

Given that we can have our pseudo-loops do what we want them to do, what sort of loop constructs might we use? We suggest the following types:

1. The "until X do . . . again" loop. If condition X exists upon entry to the loop, the loop is not executed.

2. "while X do . . . again" is similar to (1) but runs while condition X holds. If X is not true on entry, the loop is not executed.
3. "for each X do . . . loop" or "for each X (i) (for i=m to n) do . . . loop" runs the loop until the elements X are exhausted.
4. "do . . . if X then exit . . . again" exits the loop upon satisfaction of the test.

Two other control structures that are frequently used are the "if . . . then . . . else . . . endif" and the "case . . . of . . . endcase" structures. Everyone knows the first structure. We like to close each "if" with an "endif." You may or may not, depending on how you feel. The "case . . . of . . . endcase" structure may be thought of as a selection of one out of several subroutines. For example:

Structure

SHIPREC

offset	name	size	description
+0	NEXTREC	2 byte	Pointer to next record (linked list)
+2	XPOS	2 byte	X coordinate of record
+4	YPOS	2 byte	Y coordinate of record
+6	RANGE	2 byte	Distance from player
+8	VALUE	2 byte	Point value of ship

New Fighter Routine

NEWSHIP(SHIP) : Routine;

SHIP.XPOS := RANDOM(0,255); (random X coordinate)

SHIP.YPOS := RANDOM(0,255); (random Y coordinate)

SHIP.RANGE := RANDOM(MAXRANGE*3/2,MAXRANGE);
(random distance)

SHIP.VALUE := 100*RANDOM(1,3); (random value)

end NEWSHIP;

Variables and Constants

SHIP

Local reference to a SHIPREC record.

MAXRANGE

Constant giving maximum start distance for attacking fighter.

Functions and Routines

RANDOM(m,n)

Returns a random number between m and n.

Figure 3.


```

case (I) of
  (1) DOTHIS;
  (2) DOTTHAT;
  else DOTHEOTHER;
endcase

```

performs the routine DOTHIS if the selector, I, is 1; it does DOTTHAT if I=2; and it performs DOTHEOTHER for any other value of I. It is easy to implement even in languages lacking the "case" structure.

The control structures presented here should be sufficient to make the intended structure of any program clear. Often their actual implementations during coding become obscure due to language peculiarities, but a quick reference to the pseudo-code version should serve to remind you of the intended structure. Thus, you will not lose the design of your program while you are struggling with the problems of coding.

Routines and Functions

We started the design process with a loose description of what we wanted to do. We then formalized it somewhat during Level 1 by breaking the process into modules, written in structured English. From there we went on to Level 2, where we transformed the structured English to pseudo-code. At each step we assumed and used numerous subroutines and functions. So where do we describe the subroutines and functions?

There are three classes of routines. One class of routines includes those described as Level 1 modules; these, of course, we translated to Level 2 routines. We briefly described the next class of routines in the "Functions and Subroutines" part of the figures, some of them with sufficient detail to go on to pseudo-coding them. The last set are those routines that are sufficiently complex to require whole Level 1 descrip-

tions of their own. For example, a sophisticated parsing routine within another program might be sufficiently complicated to justify having its own thorough design.

By this time, you are probably pretty familiar with your design, maybe even a little sick of it. It might be reasonable to try coding a small test program to see how it looks. You may discover something that you hadn't anticipated that impacts on your design. For example, the method you selected to draw the display might prove to be far too slow. You experiment to find a faster way, but it involves a major design change. Oh well, it's better to know now before you have logged a hundred hours of coding and debugging time.

In designing a program, you should defer "programmer" problems until the final stages. But before you start the last design step, it definitely helps to consult

These are the conventions used for the various design levels in this article. They should be taken as guidelines for working out the design and are not meant to be strictly adhered to.

(a) Level 1 Conventions

Modules are worked out from the top down.

Each module is laid out in an outline format.

A control structure encompasses the lines that are indented under it (control structures usually end with an ending keyword).

Variables and data structures are only implied.

Entries should describe actions performed without referring to the "how it's done" detail.

All major activities ought to be covered, and the modules should be worked out such that all relevant cases are covered.

If any single module becomes too cumbersome, it should be redesigned or broken into smaller modules.

(b) Level 2 Conventions

Each module created during Level 1 design should be included as a routine or function. All nonprimitive routines and functions used during Level 2 design should also be pseudo-coded.

Each section of pseudo-code (each routine) should include a list of variables and constants and a list of routines and functions that were used. This provides a cross reference.

Variables and data structures should be marked as "global" if they are used in more than one routine. The actual assignment of variables to specific routines may be done during Level 3.

Variables, constants, functions, and routines are written in upper case to make them distinguishable from control words.

Semicolons are used to separate "statements" from each other. There are no rigid rules however, for statement separation.

Use nested structures. They are much cleaner than branches. Indent the lines for readability.

Use miniscule letters to indicate specific elements in data structures. They may resemble arrays, but they make it easier to specify particular elements of lists. "ELEMENT(i)" is an element of a list of elements, regardless of its actual format. "ELEMENT(I)" is an obvious array.

If you want to exit a loop prematurely, use generalized loop structures, such as:

"for each ITEM(i) do ... next (i)"

"until CONDITION do ... again"

"while CONDITION do ... again"

"do (i) . . if CONDITION then exit (i) endif ... again".

Instead of the "goto SOMEWHERE" structure, use the two conditional execution structures:

"if CONDITION then ... else ... endif"

"case (I) of (1) SUB1; ... else SUBn endcase"

(The "goto" is used only while coding, when the language lacks the sophisticated block structures or loop "exit" words.)

(c) Level 3 Conventions

The pseudo-code looks just like Level 2, except that data structures and variables are defined according to language and machine limitations.

Pointers can be initialized using the ADDR(X) function; "PTR1 := ADDR(X)" is usually pretty clear.

Reference to a location via a pointer can be coded as [PTR1]. For example, "[PTR1] := X" copies the value of X to the location reference by PTR1.

Language-supplied constructs may be used; however, it is a good idea to limit these, as this is still part of the design phase.

Figure 4.

the computer. What you learn by building a small test system will help as you struggle with the Level 3 design. Machine dependencies have a way of wrecking "machine-independent" designs just as easily as ignorance of the design can mangle your tidy code. As designer and programmer, you catch it from both sides.

Level 3 Design

We have now pseudo-coded most of the routines in our program. We know what the modules are and what they are to do. We also know about a whole mass of variables and data structures. Now we need to know what the data structures look like and how they are accessed. At Level 3, we design the data structures and the primitive operations on them.

At this point, we become concerned with the selection of the language, as well as language and machine dependencies. We should restrict ourselves, however, to dependencies relative to data structures. We will still use the techniques of Level 2 for specifying control structures and avoid implementation dependencies for these as yet; although we must address the dependencies regarding data structures, we are

still involved with design.

For this example, let's pseudo-code the NEWSHIP routine. We must also specify the layout of the fighter record SHIPREC with respect to some language. Let's assume that the language we are using permits us to reference the record and its fields by specifying a name and an offset. We first describe SHIPREC in Figure 3 (page 74).

We have specified, for purposes not connected with the NEWSHIP routine, to include a "next record" pointer in the fighter record, implying that the fighter records constitute a linked list. We have also referenced a random number generator, which we hope is at least partially supplied by the language we intend to use.

Eventually, we will have all the routines pseudo-coded in a similar manner. Figure 4 (c) (page 75) presents the Level 3 design conventions. Assuming that we are still satisfied with our design, we can go on to the final step: coding.

From Pseudo-Code to Real Code

Once we decide to start coding, we should put all our energy to writing code,

trusting that our design is complete. The actual process of coding, of course, depends upon the language we have selected. However, we might follow certain general procedures. When we embarked upon design, we worked from the top down. When coding, we work from the bottom up except for the variables, which usually must be declared prior to their use.

Generally, the transformation from pseudo-code to real code is pretty painless, for an excellent reason: you have made all the global decisions and can now concentrate on the local, line-by-line ones of coding! The loop structures, however, always seem to have it in for us. Somehow, loops always do something other than what we want them to do, forcing us to find some way of getting them to do our bidding.

As we mentioned before, most loops imply the existence of an initial element, a successor element, a final element, and some exit method. In coding a loop, then, we should find the most effective way of specifying these points. For example, in our pseudo-code we might have said, "for each SHIPREC(i) do . . ." Instead of "for I=1 to n" this might translate to:

Table of Terms

primitive

A variety of subroutine or function that performs some very basic job within the context of your program. Disk access words, I/O routines, and data structure access words might be considered primitives. Of course, if you are writing a disk access package, you might use other routines that are even more primitive.

case statement

Something like an "indexed goto," where you execute one of several possible blocks of code depending on a selector variable.

loop structure

Any program control structure that allows a block of code to be executed repeatedly. This includes such things as "TOP: (some code) GOTO TOP;" that executes the code between "TOP" and the "GOTO TOP" statement.

global variable

Technically, a variable that may be directly accessed by name from all routines in a program. For our purposes, a variable that can be accessed

by name from more than one routine. In block-structured languages, the degree of "globalness" of a variable depends upon the level of the routine within which it is defined.

data structure

Something like a collection of variables. Common data structures are: arrays, records, tables, linked lists, or files. The last type, however, usually implies disk-resident data.

module

A group of activities that can be taken together as a cohesive set. For our purposes, one of the functional subdivisions that helps describe the activity performed by the program. Modules interact with each other under the supervision of a main module.

top down design

A method of analyzing the design from the most general level, through increasing degrees of detail, until some bottom level of most intimate detail is reached. In our case, three levels of complexity occur before actual coding is reached, and most of the analysis occurs during the second level.

bottom up

The method of reconstructing the program from the lowest level of routines (the primitives) on up through the major modules. You analyze from the top down and build from the bottom up.

pseudo-code

A "fake" language designed to aid in the design process. It does what you want it to do, rather than forcing you to contort your thinking to fit its limitations.

tight (structured) English

A method of specifying an activity in a readable form. Rules for games or instructions for procedures to be followed are often written in structured English. It should get the idea across to a person, though it may not make much sense to most compilers. When you write using structured English, pretend that you are writing down a set of rules for someone to follow (as if they will be doing the activities of your program by hand).


```
PTR.S := FIRST.SHIPREC;
until PTR.S=0 do ...
PTR.S := PTR.S+0
again
```

Summary

The design process begins with an informal description of the program in terms of its actions. This is separated into modules, each of which is stated in structured English. The focus is on what the modules do, and on which one does what, rather than on how it is done. During this Level 1 design process, data structures are implied by the actions being performed and are not explicitly described.

During Level 2 design, the tight English modules are converted into routines and functions composed of calls to other routines, held together by control structures. The control structures mirror the various structures available in most languages, but they are assumed to do what we want them to do rather than what the language designer specified they should do. Loops check conditions upon

entry and may be skipped altogether if the exit conditions are already in effect. The data structures and variables implied during Level 1 are stated, but their actual formats are left vague. Data types are kept simple; integers, real numbers, pointers, characters, and strings are commonly used. You should feel free, however, to create any new data types you want.

Level 3 design starts by describing the data structures and variables used during Level 2 design. Language and machine dependencies should be taken into account at this point. Primitives that work on the data structures are pseudo-coded, still describing what they do, except that now they are designed with the language and machine in mind. We may also wish to annotate routines created during Level 2 design if they happen to directly manipulate any of the data structures.

Finally, the pseudo-coded routines are translated to the actual program code. The various routines are laid out as skeletal forms, and the variable declarations are coded. The routines are coded starting with the primitives, working up to the higher level routines. Routines should be tested as they are coded to simplify the problems associated with integration.

The transformation of loop structures, "if .. then .." structures, and "case .." structures should be watched, as language peculiarities can cause difficulties.

If you use this method conscientiously, you should be able to work out the design of a program before you get entangled in the details of coding. Coding itself should be much easier since you can ignore design considerations at that point. Working from Level 2 and Level 3 pseudo-code, even if you are working with the most primitive and unstructured of languages, you should be able to see the program structure easily by comparing the real code with the pseudo-code. Program documentation should be easier, too, as the pseudo-code provides the basis for program comments.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 194.

MATH SUBROUTINE LIBRARY



Now, a library of **Numerical Methods Subroutines** for use with your **FORTRAN** programs.

Over sixty subroutines of:

FUNCTIONS	INTERPOLATION
INTEGRATION	LINEAR SYSTEMS
MATRICES	POLYNOMIALS
NON-LINEAR SYSTEMS	DIFFERENTIAL EQ

Versions available for several FORTRAN compilers running under **CP/M-80** and **MS-DOS (PC-DOS)**.

Cost, \$250.
Manual available, \$25.

microSUB:MATH

CP/M and MS-DOS are trademarks of Digital Research Corp. and MicroSoft Corp. respectively.

503/884-3023

foehn consulting, PO Box 5123, Klamath Falls, OR 97601

FOR \$29.95, MICRO-WYL WILL WHEN OTHERS WON'T!

You'll know why INFOWORLD rated *Micro-WYL*'s performance "Excellent" when you want to insert a file into a file; when you want to print directly from the screen; when you want to see what your text will look like before printing; set tabs where you want them; write programs the easy way; and do many other things that no other word processor can do. *Micro-WYL* is a line editor; use it in conjunction with EMACS derivative word processors and you have the best of all worlds! For \$29.95, plus \$2.00 postage and handling, have it your way! Sold with the usual Overbeek 30 day money-back guarantee.

LET *Micro-WYL* DO IT ALL TODAY!

Enclosed find check for \$31.95 or

charge my Mastercard# _____ Expires: _____

Visa# _____ Expires: _____

Check format desired:

<input type="checkbox"/> 8" SSSD	<input type="checkbox"/> Osborne Single Density	<input type="checkbox"/> KayPro II
<input type="checkbox"/> Superbrain	<input type="checkbox"/> Osborne Double Density	<input type="checkbox"/> NEC 5"
<input type="checkbox"/> Northstar Advantage	<input type="checkbox"/> Morrow Micro Decision	<input type="checkbox"/> Televideo802
<input type="checkbox"/> Northstar Horizon	<input type="checkbox"/> Xerox 820 Single Density	<input type="checkbox"/> ALTOS 5
<input type="checkbox"/> Apple/Softcard	<input type="checkbox"/> Xerox 820 Double Density	<input type="checkbox"/> Epson

I'm not ready to order now, but send me information about all the affordable programs from Overbeek Enterprises.

Name _____
Address _____
City _____ State _____ Zip _____

OVERBEEK ENTERPRISES, P.O. Box 726D, Elgin, IL 60120
312-697-8420

Micro-WYL — another affordable program from Overbeek Enterprises.

Circle no. 22 on reader service card.

Circle no. 40 on reader service card.

More on Binary Magic Numbers

Edwin E. Freed presented some valuable information in his article "Binary Magic Numbers" (DDJ No. 78). Not only are the magic numbers useful, but the algorithms show how the parallel processing capabilities of even the simplest computer (in this case, the ability to process all of the bits of a "word" in parallel rather than one at a time) can be used to dramatically improve the execution time of a program.

He chose Pascal, however, to present his algorithms. I do not want to get involved with the unsolvable debate about "Which is the best programming lan-

guage?" but in this case I can confidently state that C is a better choice for the programs in the article.

The primary reason is that C has operators which map directly into machine operators, so there is no need to invent functions to mimic the hardware. The use of the operators `&`, `|`, `^`, `~`, `<<` and `>>` which stand for AND, OR, EXCLUSIVE OR, COMPLEMENT, SHIFT LEFT, and SHIFT RIGHT, makes a C version of the program much more straightforward (at least to someone who can read C). The `"op="` and `"++"` and `"--"` operators of C also correspond to the instructions available in assembly language, so I have used them.

In fact, in translating the programs to C in order to understand them more clearly, I was able to come up with cleaner versions of several of the functions presented by Freed. In most cases, I used a

more natural index to control the loops. It also turned out to be useful to split the array B into two parts. B1 is the first half of the original array — the binary magic numbers. B2 is the second half — the complement of those numbers.

The resulting functions are shown in the listing (below). I hope this will direct the attention of people back to the ideas in Freed's article. You may never need to hand-code a parity function or a side-sum function, but someday you may need to sort or sum a matrix on a multi-processor computer, and these or some other binary magic numbers may be just the trick you need in order to complete the task in log time rather than linear time.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 195.

by Dale Wilson

Dale Wilson, Codewriter, 231 Couch Ave., St. Louis, MO 63122.

Binary Magic Listing

```
/* Some useful bit manipulation functions inspired by the article
 * "Binary Magic Numbers" by Edwin E. Freed in DDJ #78, April 1983.
 *   by Dale Wilson, 1983
 *   placed in the Public Domain
 *
 * These functions were written so they may be directly translated
 * into assembly language for most computers.
 *
 * These functions were tested on a Zenith 100 computer using the
 * C86 compiler from Computer Innovations, Inc.
 */
#include <stdio.h>

#define TRUE 1 /* stranger than fiction */
#define FALSE 0 /* fiction */
#define CNTLZ 26 /* MS-DOS eof */

#define N 16 /* bits per "word" */
#define V 4 /* log 2 of N */

/* Since C does not have binary constants, the binary magic numbers are
 * expressed below in hexadecimal. B from Freed's article is divided
 * into b1 and b2 since there was no good reason to have them in the
 * same array.
 */
unsigned int b1[V] = { 0x5555, 0x3333, 0x0F0F, 0x00FF };
unsigned int b2[V] = { 0xAAAA, 0xCCCC, 0xF0F0, 0xFF00 };
```



```

/* converting binary numbers to Gray code is so simple that it may
 * be best defined as a macro rather than a function.
 */

#define binary_to_Gray(x) (x ^ x >> 1) /* X exclusive_or X shifted_right 1 */

/* MAIN routine to test the functions.
 * Numbers (entered in hexadecimal) will be used as arguments in each
 * of the functions. As an additional check, the binary number resulting
 * from the Gray_to_binary function will be converted back to Gray code--
 * which should result in the original number.
 */

main(argc,argv)      int argc; char *argv[];
{
    unsigned int r,i;
    int c;
    while(TRUE)      /* loop forever */
    {
        printf("Enter number : ");
        fscanf(stdin,"%x",&i);      /* read a hexadecimal */
        while((c=getchar()) != '\n') /* discard rest of line */
            if(c == EOF || c == CTRLZ) /* except on end of file */
                exit();      /* quit */

        printf("low clear bit: %d\n", low_clear_bit(i));
        printf("high set bit : %d\n", hi_set_bit(i));
        printf("side sum      : %d\n", side_sum(i));
        printf("parity       : %d\n", parity(i));
        r = Gray_to_binary(i);
        printf("Gray code    : 0x%04x\n", r);
        printf("GTB To Binary: 0x%04x\n", binary_to_Gray(r));
        printf("Reversed bits: 0x%04x\n", reverse_bits(i));
        putchar('\n');      /* leave a blank line between */
    }
}

/* This function returns the bit number of the lowest bit in the word
 * which is clear (zero). The least significant bit is numbered 0, the
 * bit to the left of that, 1, and so on...
 * A minus 1 is returned for words in which all bits are on.
 * The time to execute this function is proportional to V which is
 * log2 of the number of bits in a word.
 * Note that the function low_set_bit may be created by complementing the
 * argument and calling low_clear_bit.
 */
low_clear_bit(value)  unsigned int value;
{
    unsigned int temp;
    int i, result;
    if((value = ~value) == 0)      /* complement, test for zero */
        result = -1;      /* zero means no clear bits */
    else
    {
        result = 0;
        for(i = V-1; i >= 0; i--)
        {
            result <<= 1;      /* make space for next bit */
            temp = value & bit[i]; /* test using magic numbers */
            if(temp == 0)
                result |= 1;      /* next bit of result is 1 */
            else
                value = temp;      /* discard disqualified bits */
        }
    }
}

```

(Continued on next page)

Binary Magic Listing (Listing continued, text begins on page 78)

```
    return(result);
}

/* This function returns the bit number of the highest bit in the word
 * which is set (one). The least significant bit is numbered 0, the
 * bit to the left of that, 1, and so on...
 * A minus 1 is returned for words in which all bits are off.
 * The time to execute this function is proportional to V which is
 * log2 of the number of bits in a word.
 * Note that the function high_clear_bit may be created by complementing the
 * argument and calling high_set_bit.
 */
hi_set_bit(value)    unsigned int value;
{
    unsigned int temp;
    int result, i;
    if(value == 0)    /* if no bits are on */
        result = -1; /* return that info */
    else
    {
        result = 0;
        for(i = V-1; i >= 0; i--)
        {
            result <= i; /* space for next bit */
            temp = value & b2[i];
            if(temp != 0)
            {
                result != i; /* next bit is one */
                value = temp; /* discarded unneeded bits */
            }
        }
    }
    return(result);
}

/* This function returns a count of the number of bits which are on in a
 * word. It executes in a time proportional to V, the log base 2 of the
 * number of bits in a word.
 * Note that a count of the number of zero bits in the word may be found
 * by complementing the value and calling side_sum.
 */
side_sum(value)    unsigned int value;
{
    int i;
    unsigned int s;
    s = 1;
    for(i=0; i<V; i++) /* use magic in reverse order */
    {
        value = (value & b1[i]) + ((value >> s) & b1[i]);
        s <= i; /* generate the powers of two on the fly */
    }
    return(value);
}

/* This function converts a number expressed in Gray code to the
 * equivalent binary number. It executes in time proportional to the
 * log base 2 of the number of bits in the word.
 */
Gray_to_binary(value) unsigned int value;
{
    unsigned int s;
    for(s = N >> 1; s != 0; s >>= 1)
    {
        value ^= value >> s;
    }
}
```

(Continued on page 82)

GTEK

DEVELOPMENT HARDWARE/SOFTWARE
HIGH PERFORMANCE/ COST RATIO
INC. (601) 467-8048
EPROM PROGRAMMER

Compatible w/all Rs 232 serial interface port * Auto select baud rate * With or without handshaking * Bidirectional Xon/Xoff and CTS/DTR supported * Read pin compatible ROMS * No personality modules * Intel, Motorola, MCS86, Hex formats * Split facility for 16 bit data paths * Read, program, formatted list commands * Interrupt driven, program and verify real time while sending data * Program single byte, block, or whole EPROM * Intelligent diagnostics discern bad and erasable EPROM * Verify erasure and compare commands * Busy light * Complete w/Textool zero insertion force socket and integral 120 VAC power (240 VAC/50Hz available)

DR Utility Package allows communication with 7128, 7228, and 7956 programmers from the CP/M command line. Source Code is provided. PGX utility package allows the same thing, but will also allow you to specify a range of addresses to send to the programmer. Verify, set the Eprom type.

MODEL 7316 PAL PROGRAMMER
Programs all series 20 PALS. Software included for compiling PAL source codes.

Software Available for CPM,¹ ISIS,² TRSDR,³ MSDOS.⁴

1. TM of Digital Research Corp.
2. TM of Intel Corp.
3. TM of Tandy Corp.
4. TM of Microsoft.

Post Office Box 289
Waveland, Mississippi 39576
(601)-467-8048

**\$799 stand alone
MODEL 7956**

**MODEL 7956
GANG PROGRAMMER**
Intelligent algorithm. Stand alone, copies eight EPROMS at a time. With RS-232 option \$999.

**\$499
MODEL
7316**

Avocet Cross Assemblers are available to handle 8748, 8751, Z8, 6502, 680X, etc. Available for CP/M and MSDOS computers. Order by processor type and specify kind of computer.

Model DE-4 U/V Products hold 8, 28 pin parts. High quality professional construction.

MODEL 7324 PAL PROGRAMMER
Programs all series 20 & 24 PALS. Operates stand alone or via RS232.

**\$1195
MODEL
7324**

Model 7128-L1,L2,L2A \$209.00
Model 7128-24 \$289.00
DR8 or DR5 \$ 30.00
DR8PGX or DR5PGX \$ 75.00
Cross Assemblers \$200.00
XASM (for MSDOS) \$250.00
U/V Eraser DE-4 \$ 78.00
RS232 Cables \$ 30.00
8751 adapter \$174.00
8755 adapter \$135.00
48 Family adapter \$ 98.00

**\$499
MODEL
7228**

**MODEL 7228
EPROM PROGRAMMER**
All features of Model 7128 plus Auto Select Baud, super fast adaptive programming algorithms, low profile aluminum enclosure. Programs 2764 in one minute!

**\$389
MODEL
7128**

MODEL 7128 EPROM PROGRAMMER
Programs and Read:

NMOS	NMOS	CMOS	EEPROM	MPU'S
2508	2758	27C16	5213	8748
2516	2716	27C32	5213H	8748H
2632	2732	C6716	X2816	8749H
2564	2732A	27C54	48016	8741
68766	2764		12816A	8742H
68764	27128			8741H
8755	27256			8751
5133				

Circle no. 24 on reader service card.

The Best of Both Worlds

High-performance
CompuPro Hardware
with high-performance

Cromemco Software.

Cromemco's MC68000-Z80 CROMIX multi-tasking operating system with drivers to support CompuPro hardware. (requires Cromemco DPU)

Minimum configuration:

DPU - SystemSupport1 - Disk1 - Interfacer3 or 4 - 192K RAM

CROMIX with drivers to support minimum configuration \$890.
Special drivers only \$295.

The following products can be added to any 8 or 8-16 bit CROMIX system:

SCSI hard disk drivers \$195.
15Mb formatted hard disk subsystem \$2095.
30Mb (two drives) \$2995.

MDrive-H drivers \$195.
CompuPro 512K MDrive-H hardware \$1895.

Other drivers in development... custom inquiries welcome.

Authorized CompuPro Dealer.
PuterParts®

2004 4th Avenue
P.O. Box 12339
Seattle, WA 98111-4339
Telephone (206) 682-2590

OPEN
2-6 Tuesday-Saturday

"PuterParts" is a registered trademark of Katharas Companies
Additional trademarks: Z80, Zilog, MC68000, Motorola, CROMIX, DPU, Cromemco, SystemSupport1, Disk1, Interfacer3, -4, MDrive-H, CompuPro.

Circle no. 46 on reader service card.

DISK DRIVES

NEW - Full O.E.M. Warranties too!

TANDON

Model Number	Type	tpi	Height	US\$ Price
TM-100-1	SSDD	48	full	177
TM-100-2	DSDD	48	full	220
TM-101-4	DSDD	96	full	270
TM-50-1	SSDD	48	half	160
TM-50-2	DSDD	48	half	200
TM-55-4	DSDD	96	half	250
All Manuals				10
TM-848E-2	8"DSDD	48	half	380
TM-502	12.8mb Winchester			1000
TM-503	19.8mb Winchester			1100
TM-703	30.1mb Winchester			1425
300-1200 Baud RS-232 MODEM				250
Direct connect std. Auto Ans/Originate.				
Optional Auto dial/Smart Pkg.				50

ADLER COMPUTER SYSTEMS COMPANY,
125 Chandler st., Buffalo, N.Y. 14207

Phone (716) 876-1600 | NYS Residents add 7% s/s tax

Circle no. 1 on reader service card.

Binary Magic Listing (Listing continued, text begins on page 78)

```

    }
    return(value);
}

/* This function returns the parity of a word--that is, it returns a zero
 * if the number of one bits in the word is even, and a one if the number
 * of one bits in the word is odd. The low order bit of the results of
 * Gray_to_binary and side_sum both have this property, so isolating this
 * bit gives the desired result. Gray_to_binary was selected since it is
 * a faster function than side_sum.
 */
parity(value)          unsigned int value;
{
    return(Gray_to_binary(value) & 1);    /* build on previous work */
}

/* This function reverses the bits in a word. Strangly enough, this turns
 * out to be a very useful function to have available. Note that this function
 * works only on functions with word lengths which are powers of 2.
 */
reverse_bits(value)     unsigned int value;
{
    unsigned int s,i;
    s = 1;                /* s provides the powers of 2 */
    for(i=0; i<V; i++)
    {
        value = ((value << s) & b2[i]) | ((value >> s) & b1[i]);
        s <<= 1;
    }
    return(value);
}

```

End Listing

At Last! bds C ... Ver. 1.5

Including a new dynamic debugger
Still the choice of professionals

- Compiler option to generate special symbol table for new dynamic debugger by David Kirkland. (With the debugger, the distribution package now requires two disks.)
- Takes full advantage of CP/M® 2.x, including random-record read, seek relative to file end, user number prefixes, and better error reporting.
- Click option to suppress warm-boot
- New library file search capabilities
- New, fully-indexed 180 page manual
- * CP/M is a trademark of Digital Research, Inc.

V 1.5 \$120.00
V 1.46 \$115.00
(needs only 1.4 CP/M)

Other C compilers and
C related products
available ... Call!

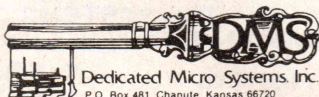
TERMS: CHECK,
MONEY ORDER, C.O.D.,
CHARGE CARD
HOURS: 9 am—5 pm
Monday —Friday
(316) 431-0018

IT'S HERE!

MONEY MATH

- Uses BCD internal representation.
- You choose from two types of rounding.
- Configurable exception handling
- Distributed with 12 digits precision. Easily configured for more or less
- Excess 64 exponents

SOURCE INCLUDED **\$5000**



Dedicated Micro Systems, Inc.
P.O. Box 481, Chanute, Kansas 66720

Huge Discounts

on Systems, Components and Peripherals

WELCOME TO ONE STOP SHOPPING: We offer a wide range of systems, peripherals and software at discounts of up to 34% off list price. Send \$4 for catalogue replete with technical specifications, and other interesting information and receive a \$10 discount coupon for your first purchase. A sampling of our discounts includes: GODBOUT 816 A \$3,682

U.S. ROBOTICS PASSWORD: \$315

HAZELTINE ESPRIT I: \$480

IMS 8000: \$3,356

NEC APC H02: \$2,758

SYSTEMS: CompuPro/Godbout, EPSON QX10, IMS International, Intercontinental Micro, Lomas Data Products, NEC APC, Seattle Computer, Systems Group, Tarbell Electronics.

PRINTERS: Epson, Mannesman Tally, NEC Spinwriters, Texas Instruments, Teletype Corp.

GRAPHICS: Autocad Software, Houston Instruments Digitizer and Plotters, Scion MicroAngelo.

PLUS: CORVUS AND QCS WINCHESTERS, U.S. ROBOTICS MODEMS, SOFTWARE, 3M DISKETTES, IBM 3270 NETWORK TERMINALS AND PRINTERS, QUADBOARD.

*** WE EXPORT TO ALL COUNTRIES ***

JOHN D. OWENS ASSOCIATES, INC.

DEPT. DD 12 SCHUBERT STREET
STATEN ISLAND, NEW YORK 10305

(212) 448 6283 (212) 448 6298 (212) 448 2913

TWX: 710 588 2844 CABLE: OWENSASSOC.

Prices subject to change without notice. We have no reader inquiry number.

A Sample Run:

Enter number : 0001
low clear bit: 1
high set bit : 0
side sum : 1
parity : 1
Gray code : 0x0001
GTB To Binary: 0x0001
Reversed bits: 0x8000

Enter number : 0002
low clear bit: 0
high set bit : 1
side sum : 1
parity : 1
Gray code : 0x0003
GTB To Binary: 0x0002
Reversed bits: 0x4000

Enter number : 0003
low clear bit: 2
high set bit : 1
side sum : 2
parity : 0
Gray code : 0x0002
GTB To Binary: 0x0003
Reversed bits: 0xC000

Enter number : 0004
low clear bit: 0
high set bit : 2
side sum : 1
parity : 1
Gray code : 0x0007
GTB To Binary: 0x0004
Reversed bits: 0x2000

Enter number : 8000
low clear bit: 0
high set bit : 15
side sum : 1
parity : 1
Gray code : 0xFFFF
GTB To Binary: 0x8000
Reversed bits: 0x0001

Enter number : 7FFF
low clear bit: 15
high set bit : 14
side sum : 15
parity : 1
Gray code : 0x5555
GTB To Binary: 0x7FFF
Reversed bits: 0xFFFE

Enter number : FFFF
low clear bit: -1
high set bit : 15
side sum : 16
parity : 0
Gray code : 0xAAAA
GTB To Binary: 0xFFFF
Reversed bits: 0xFFFF

Enter number : 1111
low clear bit: 1

Enter number : 0000
low clear bit: 0
high set bit : -1
side sum : 0
parity : 0
Gray code : 0x0000
GTB To Binary: 0x0000
Reversed bits: 0x0000

high set bit : 12
side sum : 4
parity : 0
Gray code : 0x1E1E
GTB To Binary: 0x1111
Reversed bits: 0x8888

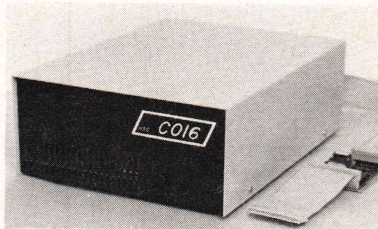
Enter number : 3333
low clear bit: 2
high set bit : 13
side sum : 8
parity : 0
Gray code : 0x2222
GTB To Binary: 0x3333
Reversed bits: 0xCCCC

Enter number : 7777
low clear bit: 3
high set bit : 14
side sum : 12
parity : 0
Gray code : 0x5A5A
GTB To Binary: 0x7777
Reversed bits: 0xEEEE

Enter number :

End Sample Run

SAVE YOUR 8 BIT SYSTEM Join the 16/32 Bit Revolution Through Evolution, For Under \$600



Would you like to run CPM86™, CONCURRENT CPM86™, CPM68K™, MS-DOS™, IBM™ PC APPLICATIONS, DEVELOP and TEST 16/32 BIT SOFTWARE, or add an INTELLIGENT HIGH SPEED RAM DISK to YOUR SYSTEM?

You can with the HSC CO16 ATTACHED RESOURCE PROCESSOR:

CO16 with either 8086, 80186, or 68000 16 bit micro-processor and up to 768K Bytes of parity checked RAM may be connected to virtually any Z80 based computer system or APPLE 2E computer system.

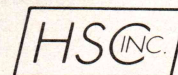
APPLE 2E™
RADIO SHACK TRS80™
ZENITH™
OSBORN™
ALTOS™
KAYPRO™
COLUMBIA™
TELEVIDEO™
DEC™
or OTHER Z80 BASED
SYSTEM

An 8 bit system equipped with CO16 retains all of its original capabilities PLUS it has the added ability to run most 16 bit operating systems and applications.

Prices start at \$595.00 which includes the CO16 processor with 128K bytes of memory, two volume user manual, and the HSC Software Development System. Hardware options include memory expansion to 768K bytes, attractive desk top enclosure with power supply, and the INTEL™ 8087 Math Co-processor (available on CO1686 only). Software options include CPM86, CPM68K, Concurrent CPM86, or MS-DOS™ operating systems. The UNIX™ operating system will be available in the second quarter of 1984.

For Information on CO16 Contact

HSC INC.
Box 86
Herkimer, NY 13350
(315) 866-2311



Dealer and OEM inquiries are invited.

Circle no. 25 on reader service card.

SXR PLUS: Sorted Cross Reference

Company: Prasek Computer Systems, Inc.,
Box 2427, Santa Clara, CA 95055

Computer: Apple II/II+ (48K RAM) or
Ile with DOS 3.3 and Applesoft One
disk drive; *Optional (but strongly
recommended):* Printer, 80-column
card, second disk drive

Price: \$39.95

Circle Reader Service No. 133

Reviewed by Charles Petzold

SXR PLUS is a handy utility for programmers who need a cross-referenced index of variables in an Applesoft program. It is especially useful when debugging or modifying a program.

SXR PLUS provides 40- or 80-column output to a screen or printer. Output may be tailored to include or exclude line number references, numeric constants, and/or quoted literals (strings). A search feature is included to find specific variables. Special instructions are provided for programmers who want to move DOS into the upper 16K of an Apple Ile or to a 16K RAM card on an Apple II+.

Complete instructions are contained in a 24-page booklet. It's slow going the first time through because Prasek takes great pains to make sure you understand every step. After that first run-through, however, the procedure is so simple the manual isn't needed unless you start having problems. If that happens, a troubleshooting guide is included to put you back on track.

When SXR PLUS is booted, a menu-driven program is used to set initial parameters. You must decide whether you want line number references, numeric constants, and/or quoted strings included in the cross-referenced list. If you choose standard video output (a monitor or TV set), 40-column width is automatic. If you choose another output device (such as a printer), you must select the proper slot number and determine whether you want 40-column or 80-column output. The output will pause every 20 lines if you select the pause option.

Once the parameters are set, the SXR PLUS diskette is removed from drive 1 and replaced with the disk containing the program to be cross referenced. After this program is loaded, the disk is removed and the SXR PLUS disk reinserted. To get a cross reference, BRUN SXR PLUS and select either F for a full sort (as determined by the parameters) or S to search

for a specific variable. That's about all there is to it.

SXR PLUS comes in handy when you're debugging. Good programmers keep a list of every variable they use. If you make a typo and accidentally use BA\$ instead of AB\$, BA\$ will show up in a sorted cross reference, and you will have found the error and the line number where the mistake was made.

Another use for SXR PLUS is when you decide for one reason or another to change a variable. If you don't change every occurrence, you've got trouble. Using the search function, you'll be able to find (and later change) every occurrence.

Depending on the complexity of the program, a sort may take only seconds or it may take several minutes. For test purposes, I used the 47-sector LEMONADE program from Apple; a full sort took more than six minutes. I also tried a 91-sector program (PLANET.OF.THE.ROBOTS on a Softdisk Magazine), and a full sort took more than eight minutes. Both sorts were viewed on the monitor, not a printer.

Although SXR PLUS did everything it promised, there were some minor annoyances. To set the initial parameters, for example, you must answer six or eight questions, depending on your answers. After answering all questions, you are asked if the parms are correct. If they are not, you must repeat your answers to *all* questions. It would be quicker and easier to change only those answers that need correction (or updating). This can be easily fixed with some minor programming changes.

Another nice feature would be default parameters, with the default option being the option the user would most often select. To accept the default option, the user would merely hit the RETURN key.

Also, no provision is made for a two-drive system. With some more minor programming changes, this could be added. It would eliminate the disk-switching required under the one-drive system.

Since SXR PLUS is not on a protected disk, experienced programmers will no doubt be quick to make the necessary modifications.

With respect to copies and distribution, Prasek has what they call a "share-the-fare" program. You pay \$39.95 for the first SXR PLUS package, and \$7.50 for each additional package. There is no limit or restriction on the number of additional copies you can purchase at this

price. User manuals alone are now \$6.00.

If you destroy your disk, you can also get a replacement disk for \$5.00 if you return the original disk. While the old warranty was only five days, you are now protected for 90 days. Returns are made to Prasek.

Perfect Writer Perfect Speller

Company: Perfect Software, Inc., 1001
Camelia St., Berkeley, CA 94710

Price: \$399.00

Circle Reader Service No. 135

Reviewed by Charles K. Ballinger

By now you probably have read innumerable reviews on the most common word processors currently on the market and still can't decide which one is for you.

Well, perhaps you are going about it the wrong way. As in most things you purchase, don't you decide what you want the product to do and then go out and find something that fits your requirements? (Of course, you know full well that in all things compromises must be made.)

It has been 5 months now since I first received this software for review. I have run it through its paces enough to feel comfortable with it, and I can now tell you what may make you interested in this product.

Since I do a lot of programming in a variety of languages, I look for a word processor that can handle the demands made on it by a special group of users (programmers) and still be used to produce the documentation that is so necessary for an item I would produce. The following are things I wanted in a word processor but had not been able to find in any of the others currently on the market:

1. No need for imbedded special codes that would prevent me from sending text files to other systems.
2. Ability to view more than one input file in order to merge more than one program source into a new program.
3. The absence of memory restrictions on the size of the source that I could edit at one time.
4. The ability to produce documentation that consists of several files and to print them in a contiguous fashion and also produce more than one copy at a time.

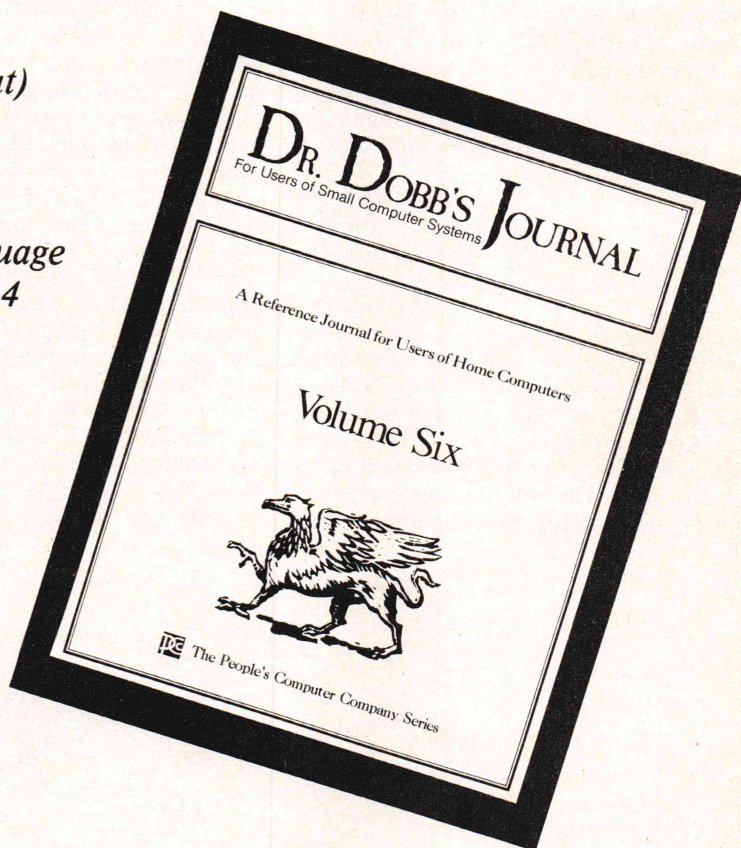
I found that Perfect Writer lives up

Volume 6

Now Available!

At last what you've been waiting for: **ALL** the issues from 1981 in one **GIANT** volume—over 550 pages long!

- *The first all-FORTH issue (now sold out)*
- *More Small-C development*
- *Continuing coverage of CP/M*
- *Santa Barbara Tiny BASIC for 6809*
- *Pidgin - A Systems Programming Language*
- *Write Your Own Compiler with META-4*
- *The first "Clinic" columns*
- *The Electric Phone Book*
- *J.E. Hendrix's Small-VM*
- *What FORTH Is?*
- *Several exciting Z80 utilities*
- *The Conference Tree*
- *PCNET information*
- *Several North Star tidbits*
- *An assortment of utilities*
- *And much, much more*



YES! ☐ Please send me the following Volumes of Dr. Dobb's Journal.
☐ ALL 6 for ONLY \$125, a savings of over 15%

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Card# _____ Expiration Date _____

Please charge my:

- ☐ Visa ☐ American Express
☐ MasterCard ☐ I enclose ☐ Check/Money order

This offer expires
March 31, 1984.

*These are reduced prices
for Dr. Dobb's readers.*

Vol. 1	_____	×	\$23.75	=	_____
Vol. 2	_____	×	\$23.75	=	_____
Vol. 3	_____	×	\$23.75	=	_____
Vol. 4	_____	×	\$23.75	=	_____
Vol. 5	_____	×	\$23.75	=	_____
Vol. 6	_____	×	\$27.75	=	_____
All 6	_____	×	\$125.	=	_____

Subtotal _____

Postage and Handling _____

Must be included with order.

*Please add \$1.25 per book in U.S.
(\$2.00 each outside U.S.)*

MAIL TO: Dr. Dobb's Journal P.O. Box E, Menlo Park, CA. 94026

Circle no. 72 on reader service card.

L8

TOTAL _____

As supplied (on 8-inch disk), the software was fairly easy to install. Perfect Writer is written in C and, as such, provides the authors a means to continually update the product. Prior to installation you should read the introduction to get an overview of the software. Pay close attention to the information on swapping (this is how Perfect Writer allows files larger than available memory), because this is where your disk capacity may play an important part. After the introduction you can proceed to the installation guide in Appendix A.

If you can't find your terminal type on the menu then you may go through a question/answer session with the program to arrive at the correct terminal configuration. This is one of the first programs I've seen where they make it perfectly clear just what you are supposed to respond with. The questions that require other than a yes or no response tell you explicitly to enter your response in hex or in decimal as required, instead of leaving it up to you to guess whether you enter hex, decimal, octal, or ASCII as a response.

The documentation may, at times, seem overwhelming, particularly when you first start to use the software. Your best approach would be to skim the material and then start with the lesson disk to get an actual feel of the programs.

The associated fold-out reference card is very handy and goes a long ways in helping you find that command you are looking for. All in all their documentation rates a "well done."

disk, it is almost impossible not to gain a working familiarity with Perfect Writer/Speller in a couple of hours. Granted you won't know all the commands or capabilities, but then of what value is a software product that you can take to its outer limits in such a short period of time? I'll tell you, it's probably one that you will replace very soon. Most of the commands have a scheme that is quickly apparent, and you'll find that they are easily memorized. What's nice about Perfect Writer/Speller is that I'm still discovering things that it can do.

Perfect Speller contains a 50,000 word dictionary which is far easier to use than MicroPro's SpellStar. The speed, at least on my machine (2 MHz 8080 with 8-inch drives), approaches the estimate given in their manual: approximately 4000 words/minute.

I only had one minor problem with Perfect Writer/Speller (my own mistake, caused by being overzealous). When I contacted Perfect Software, Inc. — they even talk to users, unlike some — I found them to be most helpful and willing to provide the assistance I needed, pointing out that I had inadvertently missed a step. Not having identified myself as a software reviewer, I can only conclude that this is the normal support level that all users can expect to receive.

As mentioned earlier, this product is a true help to a programmer when it comes to constructing new programs from existing ones. The ability to have multiple file buffers is ideal from a programming point of view because you can now pull currently running source code into your new creation without undue effort. In fact, you are allowed access to a maximum of seven input documents, and, with the ability to use split-screen mode, new program construction time should be cut by a considerable margin.

All things considered I find this a perfect programmers' tool. It provides you with a product that will assist you from coding through final documentation of your software. Now you only have to learn one product to produce both your source code and your finished documentation, without the fuss and bother of trying to remember more than one set of word processing commands. I'm sure you have had the problem of mixing commands if you currently use more than one word processor or have access to different machines.

I heartily recommend this product as an excellent programming tool and documentation aid. With the introduction of the additional Perfect software family, and the trend of software vendors to provide integrated software, I think this is going to be a hard product to beat.

EXPAND YOUR HOME COMPUTER LIBRARY WITH THESE BOOKS FROM RESTON/PRENTICE-HALL AND DR. DOBB'S JOURNAL

Programming Languages

The C Programming Language by Brian W. Kernighan and Dennis M. Ritchie

Prepares you to make effective use of the C language. This book gives you detailed explanations and examples of all the features of the language. You get a full description of the I/O library and learn how to write programs that will be portable from one system to another without changes. You also learn the UNIX operating system interface.

1978 228 pp. cloth/\$15.95 Prentice-Hall

The C Puzzle Book, Puzzles for C Programming Languages by Alan R. Feuer

A perfect companion to The C Programming Language Book, this puzzle book helps you understand C syntax and know what meaning a translator will ascribe to properly formed constructions. Each programming puzzle comes with its solution and a step-by-step explanation of how the solution is derived. These puzzles challenge your mastery of the rules of C and exhibit pitfalls in the language that are most often learned only through trial and error.

1982 173 pp. paper/\$12.95 Prentice-Hall

Pascal for the Apple by Ian MacCallum

For the computer enthusiast who wants to learn how to program in Pascal on the Apple II and Apple II Plus. Leads you line-by-line through sample programs. Uses attractive graphics and the top-down approach to programming. You learn methods of designing correct programs. A supplementary disk of programs is also available to run as you study the book.

1983 500 pp. cloth/\$16.00 Prentice-Hall

Operating Systems

A UNIX Primer by Ann Nichols Lomuto and Nico Lomuto

This beginning level introduction to the UNIX system is written entirely from the user's problem-solving viewpoint. After just 15 minutes of reading, you'll be able to start using UNIX—by the end of the book you'll be a sophisticated user. Realistic examples illustrate why features exist (rather than just how to use them). You'll appreciate the imaginative, practical applications and troubleshooting tips this hands-on book offers.

1983 240 pp. cloth/\$19.95F Prentice-Hall

The Programmer's CP/M Notebook: Applications and Analysis by David Cortesi

Displays the process of designing and building software for the CP/M operating system. Contains 12 complete, functional programs of considerable length, which, if purchased from software publishers, would cost between \$100 and \$300. For each program, the book considers your needs as a user, develops a verbal specification, works out the program logic in a high-level language, and presents the assembly language code that implements the program.

1982 352 pp. cloth/\$15.95 Reston
paper/\$10.95

Starting FORTH by Leo Brodie

Gives you a clear and comprehensive introduction to FORTH, the revolutionary approach to computer programming. FORTH increases your control over your computer and environment. This book has everything from the basics to advanced topics such as combining words, vectored execution and FORTH techniques for fixed-point arithmetic through scaling.

1981 384 pp. paper/\$15.95 Prentice-Hall

Using the UNIX System by Richard Gauthier

A down-to-earth guide to the many program development features of the UNIX system. This is an excellent handbook that shows you how to handle everything from specific commands to files to overall system design for new applications. It's a valuable addition to everyone's professional library.

1981 297 pp. cloth/\$18.95 Reston

Software Engineering

Designing Systems with Microcomputers: A Systematic Approach by M. David Freedman and Lansing B. Evans

Here's a self-contained method for designing systems that contain microcomputers. You get cost-effective engineering approaches for designing, constructing and integrating software and hardware. There are ingenious examples of implementation, such as how to design a microcomputer-based burglar alarm system. All design procedures are given step by step with extensive documentation.

1983 320 pp. cloth/\$26.00F Prentice-Hall

Robotics

Robotics and Automated Manufacturing by Richard C. Dorf

This unique book considers the fundamental concepts and applications of robotics and computer-aided manufacturing systems that may be effectively utilized in the nation's work places. One of the first thorough examinations of this exciting new field, this book considers the issues of productivity and automation, and then reviews the history, development and classification of robots, their mechanical and electrical components, and sensors and vision systems. Also addressed are economic, labor, and work place considerations and the future of robotics and automated manufacturing.

1983 208 pp. cloth/\$24.95 Reston

Computer Education

Computer Anatomy for Beginners by Marlin Ouverson

A perfect guide to home computers for the absolute beginner. If you're curious about what computers do, but don't know what questions to ask—this book is written for you. It answers all your questions: How to decide which computer to buy, whether you really want a computer or not, what uses you are likely to find for it, where to get assistance, and what questions to ask.

1982 208 pp. paper/\$10.95 Reston
cloth/\$15.95

Mail coupon to: Dr. Dobb's Journal/P.O. Box E/Menlo Park, CA 94026

Yes! Please send me the book(s) in the quantities I've indicated below:

- ___ A Unix Primer, 93888-6 cloth @ \$19.95F
- ___ Computer Anatomy for Beginners, paper R0919-2 @ \$10.95
cloth R0920-0 @ \$15.95
- ___ Designing Systems with Microcomputers, cloth 20135-0 @ \$26.00F
- ___ Pascal for the Apple, paper 65289-1 @ \$16.00
- ___ Programmer's CP/M Notebook, paper R5641-7 @ \$16.95
cloth R5642-5 @ \$21.95
- ___ Robotics and Automated Manufacturing, cloth R6686-1 @ \$24.95
- ___ Starting FORTH, paper 84292-2 @ \$15.95
- ___ The C Programming Language, cloth 11016-3 @ \$15.95
- ___ The C Puzzle Book, paper 10992-6 @ \$12.95F
cloth 10993-4 @ \$16.95F
- ___ Using the UNIX System, paper R8162-1 @ \$15.95
cloth R8164-7 @ \$21.95

Enclosed is my payment of _____ in check or money order.
☐ VISA ☐ MASTERCARD (Interbank # _____) ☐ American Express
Card No. _____
Expiration Date _____

Signature _____
Name _____
Title _____
Organization _____
Address _____
City _____ State _____ Zip _____
Phone (____) _____

Please allow 4-6 weeks for delivery

Add \$1.50 per book for shipping and handling charges

Cryptography. Proceedings of the Workshop on Cryptography, Burg Feuerstein, Germany, Mar 29-Apr 2, 1982.

Edited by Thomas Beth
Published by Springer-Verlag, 1983
\$18.50, 402 pages
Reviewed by Richard L. Lozes

A Proceedings volume must be viewed from two perspectives: global and detailed. A smooth, logically connected, and well-styled global view is the responsibility of the editor. An accurate and complete paper is the responsibility of the authors.

These Proceedings cover a wide range of topics from classical cryptography to the Rivest-Shamir-Adleman scheme. Given the strides made in cryptography in the last decade, it is surprising to find papers concerning mechanical contrivances such as the Enigma machine. This historical view, nonetheless, is helpful; it motivates the abstract mathematical arguments later in the volume by means of physical analogies. The editor has performed his job ably.

The authors, too, hold up their end of the bargain. By and large, the papers are well organized and clearly presented. Understandably, one or two authors have difficulty with the English language.

Certainly, a Proceedings volume can never be recommended as a self-study text. However, I suspect that students of cryptography, with little prior exposure, could glean much valuable knowledge, especially if they were careful to study and commit to memory the introductory mathematics to be found in the first paper.

IBM Data Files

By David Miller
Published by Reston Publishing/Prentice-Hall
\$15.00 paper, 260 pages
Reviewed by James Moran *

One of the failures of documentation for personal computers is the paucity of adequate information on the use of disk

files. Documentation for the IBM PC is not an exception, unfortunately, but puzzled PC users might find some needed answers in this book.

In *IBM Data Files*, author David Miller has created a useful tutorial for novice users of the IBM PC. The book assumes that the typical PC owner's familiarity with disk devices stops with the user's ability to load a piece of packaged software, and it attempts to address that limited knowledge in classic textbook format. Information is presented clearly in a precise manner and is followed by short quizzes to test the reader's understanding. One of the nice touches in the book is the grouping of programming examples at the end of each chapter. After finishing a chapter, the reader keys in the examples and runs them on the PC. In addition to reinforcing the learning process, this concept leaves the reader with a library of practical and usable programs after the book has been read.

IBM Data Files begins with a short introduction on the mechanics of using a diskette drive and progresses to file design, programming techniques, and, finally, file planning strategies for integrated systems — a particularly useful section, if a bit short. VisiCalc users will find the section on creating DIF files to be the most useful part of the book since the author was most generous in furnishing examples on how to standardize user file formats so that they may be transferred for use in a VisiCalc program.

This book does have some minor weak spots, and those would be most noticeable to readers towards whom author Miller seems to have directed his information: the beginning user. For example, it is somewhat surprising not to find a single diagram or graphic to visually enlighten the reader about the physical organization of a diskette. Some examples of random access algorithms would also have made this book more useful to PC owners who have an intermediate level of knowledge about their equipment.

All things considered, the book would probably be most useful to beginning and intermediate users. Those who are more technically knowledgeable may prefer a book with greater depth or more advanced topics although even they might find that the working program examples are worth the price of the book. Among those programs is a complete home inventory system that is contained in 36 pages of BASIC code. And for those readers that

would rather not spend more than a few hours keying in the programs, an offer by the author to supply the programs on diskette for fifteen dollars seems like a pretty good deal.

Microprocessor Support Chips: Theory, Design, and Applications

By T. J. Byers
Published by MicroText/McGraw-Hill,
August 1983
\$38.00 hardcover, 224 pages (170 illustrations)
Reviewed by David W. Carroll

Microprocessor system designers have recognized that in most applications it is neither practical nor efficient to have a microprocessor perform all of the routine tasks required in a complex system. Rather, specialized support integrated circuits (ICs) have been developed to handle individual system support requirements.

Recognizing that these parts are necessary for economical, high performance designs in today's complex microprocessor-based systems, T. J. Byers has compiled a collection of 97 state-of-the-art support chips in his book, *Microprocessor Support Chips: Theory, Design, and Applications*, recently published by MicroText/McGraw-Hill. This design reference offers often hard-to-get information on many support ICs, including pin-outs, typical application schematics, and specific interfacing and design information. This book is not intended to replace the manufacturers' data sheets; rather it supplements them with real-world application information.

Some areas covered include telecommunications, power supply and special purpose, interface, control, video, and A/D and D/A converters. Telecommunications parts include various serial communications interfaces, protocol converters, local area network controllers and interfaces, and modem chips. Under power supply and special purpose we find keyboard and display controllers, data encryption chips, timers, parallel interfaces, and pulse width modulators for switching power supplies. The control ICs section includes printer and stepper motor controllers, floppy disk formatters and controllers, and Winchester hard disk controllers. The video chapter covers most currently popular

*Copyright © 1983 by Compu-Syn.

Overall, *Microprocessor Support Chips* lives up to its author's stated goal to "greatly facilitate the use of new chips in current designs." However, I find its subtitle, *Theory, Design, and Applications*, misleading. Very little theory is presented and design aid is mostly by example. Perhaps *Design Applications* would have been a more appropriate subtitle for *Microprocessor Support Chips*.

By James W. Coffron
Published by SYBEX
\$15.95, 295 pages
Reviewed by Chuck Ballinger

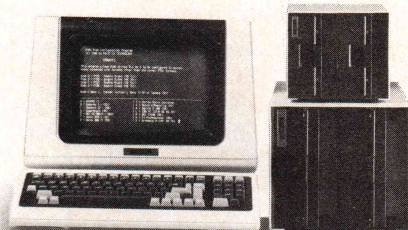
The book first takes you on a tour of the various ROMs (ROM, PROM, EPROM, and EAROM) and gives an in-depth explanation of each one in terms of function and application. Continuing on the tour, both static and dynamic RAM are explained, along with a discussion of circumstances where one would be preferred over the other. Complete layouts are also presented for most of the cur-

With the current interest in modern communications, the chapter on serial communications may help you explore the hows and whys. Start bits, parity bits, and stop bits are described clearly and thoroughly. For communications

The discussions of interrupts and programming the command modes for the serial I/O chips make the book a valuable addition to anyone's library. With the advent of newer CPU's many say the 8080 and Z80 will disappear. To the contrary, I say that you will start seeing the devices in your toasters, blenders, etc., and perhaps you would like to know what makes

PT2x — Double Sided 48 ■ TeleVideo 802 ■ Associate/ Gnat ■ R
Superbrain Quad ■ BMC if800 ■ Sanyo ■ IBM PC/ TI Pro (CP/M-86)
■ Morrow MD3 ■ Telcon (Zorba/ Nomis) ■ Olivetti M20 ■ NCR Decision
820-II ■ RAIR Black Box ■ Toshiba T-100 PC ■ HP 86A ■ Micro Source M
KayPro IV ■ PT2x — Single Sided 48 ■ KayPro II ■ Osborne I ■ Osborne II
(8001A/ 8031) ■ Xerox 820-I ■ Xerox 820-II ■ TRS-80 Mod I ■ Morrow Micro
W/ Magnolia ■ Zenith Z-100 ■ IBM PC/ XT Pro ■ HP 86A ■ Micro Source M
■ Access ■ Zenith Z-100 ■ Hantek 9100 ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro
■ Basic 4/ AOS ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro
Discovery
Digilog ■ Sanyo MBC 1250 ■ ARC Datcher Micro ■ Northern Telecom 303 ■ Seiko 8
Sing ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro
Sided ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro ■ IBM PC/ XT Pro
Xerox 820-I ■ Xerox 820-II ■ Xerox 820-I ■ Xerox 820-II ■ Xerox 820-I
MBC 5000 ■ PT2x — Single Sided 8" ■ CP/M 555D (IBM 3740) ■ Xerox 820-I ■ TRS-80
and Trout ■ SD Systems SBC-200 ■ Altos ■ CompuPro System 8/16 ■ Xerox 820-II
(CP/M) ■ CCSS/ Morrow ■ PT2x — Double Sided 48 ■ TeleVideo 802 ■ Associate/ C
■ Otrona ■ Heath Z-90 ■ Superbrain Quad ■ BMC if800 ■ Sanyo ■ IBM PC/ TI Pro
Wangwriter ■ Epson QX-10 ■ Morrow MD3 ■ Telcon (Zorba/ Nomis) ■ Olivetti M20
■ Zenith Z-100 ■ Xerox 820-II ■ RAIR Black Box ■ Toshiba T-100 PC ■ HP 86A ■

CP/M is a Registered Trademark of Digital Research



DISCANT

89

them tick. This book can be used both as a reference manual and a repair manual since it gives you an in-depth working of what goes on under your CPU's cover.

Concepts for Distributed Systems Design

By Gregor von Bochmann

Published by Springer-Verlag, Berlin-Heidelberg

\$19.00, 259 pages

Reviewed by Robert Irving

Von Bochmann uses very formal guidelines when writing about systems. A full half of the book, including the annexes, is devoted to the requirements for specification of a distributed system. The first quarter defines and gives examples of distributed systems, specifically excluding multiprocessor systems but including a modular system in one physical location, which is laid out so that it could be distributed. The second quarter reviews architecture and communications protocols.

The reader is assumed to be a computer professional well-versed in conventional computer architecture and modular software design. This approach, combined with the terse, formal text and lack of an index, limits the application of this book as a text as well as a reference volume. However, if you already have a copy of Paul Green's *Computer Network Architectures and Protocols* (Plenum 1982), this book would be a useful supplement with regard to writing system specifications.

An Introduction to Numerical Methods with Pascal

By L. V. Atkinson and P. J. Harley

Published by Addison-Wesley Publishing Company

\$16.95, 300 pages

Reviewed by Robert Ashworth

This book blends numerical methods with Pascal and calls upon the benefits of those data types to provide a natural implementation of the respective methods. While it is a college text for those of a scientific bent, no knowledge of numerical analysis is required. Familiarity with Pascal is assumed, and the text flows very well and can quickly be adapted for micros or even mainframe environments.

The text begins with a review of Pascal and builds on its merits: transparency (the intention of a well-written program being self-evident), security (the compiler helping to detect errors), and efficiency

(its design taking implementation into account). Emphasis is on the implementation of methods on a computer so as to reinforce their understanding.

Next comes a full treatment of rounding errors: those contributed by the method selected, and those inherent in the computer itself. Chapters 3-5 cover the solution of linear and nonlinear equations. Here the LU decomposition techniques are given prominence and the inverse of a matrix is explained. This part concludes with the treatment of the eigenvalue and the eigenvector problems, as well as a short discussion of the non-symmetrical cases.

The next three chapters deal with discrete functions, differentiation and integration, and finally present an overview of ordinary differential equations. The Pascal programs are written out in full, extending over several pages.

Exercises were provided for each chapter but a solutions appendix was not included. I found the text to be very well written, with sixty-two useful programs.

Augmented Transition Networks

Edited by Leonard Bolc

Published by Springer-Verlag, 1983

\$29.00, 213 pages

Reviewed by Chuck Ballinger

Confused by the title? Well, if you are not sure what an Augmented Transition Network is I'll try to explain. An Augmented Transition Network (ATN) is used in the design of interpreters, compilers, and editors as a method of factoring the input. Assume you have a data base that contains information on airplanes and you have presented a computer terminal with the following question: "How many Skyhawks required engine repair in 1973?" The processing of that request would require three main steps: parsing, interpretation, and evaluation. The first phase is what this book, as a collection of four papers, attempts to explain.

As a collection of papers that are related, but not directly integrated, there is some discord in the authors' presentations. Due to the nature of the subject, the amount of deviation does not detract from the initial subject, provided you are well versed in ATNs before picking up this book. This is not a tutorial or step-by-step book. If you don't know ATNs at the onset this book will not help you pick up any additional information.

Unless you are well versed in compiler design and implementation this book will be far above the average- or medium-level programmer. This text is intended for a very limited audience with a very specialized background or interest level. Since it

is a collection of four papers, and each paper has little cross-relationship with the other, you must understand the material from the onset in order to have any inkling of what the individual authors are trying to convey.

If you are involved in the design of compilers, interpreters, or editors then this book may be of value to you. The textbook value may justify the cost as the areas covered include, "The Planes Interpreter and Compiler for ATN Grammars," "An ATN Programming Environment," "Compiling ATN into MacLisp," and finally "Towards the Elastic ATN Implementation." If one of these individual papers is of interest to you then you might be interested in this book. For the vast majority of computer users/programmers this book should be bypassed. For those interested, the book is full of examples of the various methods of ATN. The entire text is double-spaced in typical college manuscript format and is well structured.

DDJ

Letters

(Continued from page 7)

Fast Conic Curves

Dear DDJ:

A general algorithm for drawing any conic curve at any orientation uses a slightly different viewpoint than the one detailed by Michael Enright (DDJ No. 86, December 1983, pp.19-20), but still yields all the benefits of his method. Assuming the cursor starts on the curve, the best direction of the four possible directions is chosen by evaluating the potential distance that each would be from the actual curve and taking the one that gives the smallest. This distance is found by using a form of the gradient which for conic equations is linear, i.e., using addition not multiplication. By updating a table of possible changes again using only addition, the gradient at each point can be maintained. The algorithm continues in this way until it nears the endpoint. The algorithm draws the "best possible" curve — the one closest to the actual curve.

I developed this method three years ago for drawing circles and circular arcs. In the degenerate case it yields the standard method for drawing lines. With the growing speed and capacities of microprocessors, I believe that algorithms that "look before they leap" will be increasingly useful.

Jim Hatton
3715 Summit Drive
Mt. Shasta, CA 96067

DDJ

Q/C

For only \$95, Q/C is a ready-to-use C compiler for CP/M. You get complete source code for the compiler and over 75 library functions. Q/C is upward compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, and bit fields.

- Full source code for compiler and library.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Excellent support for assembly language and ROMs.
- Q/C is standard. Good portability to UNIX.

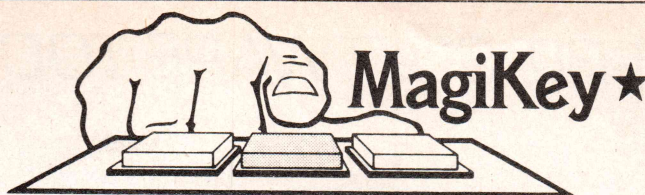
Version 3.2 of Q/C has many new features: structure initialization, faster runtime routines, faster compilation, and improved ROM support. Yes, Q/C has casts, typedef, sizeof, and function typing. The Q/C User's Manual is available for \$20 (applies toward purchase). VISA and MasterCard welcome.

THE CODE WORKS

5266 Hollister
Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

Circle no. 9 on reader service card.



THE FULL-FEATURED KEYBOARD EXPANDER for all 8080-8085-Z80 computers using CP/M 2.2

MagiKey™ will redefine your keys as character strings ... and transform single keystrokes into commands for programs, words for word processors, data for data bases, and messages for modems. Without hardware or system modifications.

MagiKey™ has more advanced features than any other CP/M keyboard expander. For example:

- * Redefine a key to send the string:
"Run WordStar and edit form letter number 17"
Hit the key. YOU will see the string, but "WS FRMLTR17" will be sent to CP/M. MagiKey™'s CONSOLE REDIRECTION can display messages which are invisible to programs and CP/M ... very useful for recalling key assignments, custom operator prompts, and making CP/M friendlier.
- * Redefine a key to run several programs in sequence. Each program can wait for keyboard input or receive pre-defined commands and data. MagiKey™'s built-in BATCH PROCESSING doesn't use CP/M's SUBMIT, and handles programs that CP/M's XSUB can't.
- * Redefine a key to display the prompt:
"Execute SuperCalc using the spreadsheet file"
Press the key to display the prompt. Type the file name, hit RETURN, and you're into SuperCalc. When done, a single keystroke saves your updated spreadsheet. You don't have to remember or retype its name. MagiKey™'s RECURSIVE key redefinition mode automatically does it for you.

WE INVITE COMPARISON

\$100

8" SSSD, most 5 1/2" formats
add 6% tax in CA
check, VISA, M/C

CP/M (tm) Digital Research
SuperCalc (tm) Sorcim
WordStar (tm) Micropro

PRO

microSystems

16609 Sagewood Lane
Poway, California 92064
(619) 693-1022

Circle no. 44 on reader service card.

PROGRAMMERS-

COMPARE Products. Get Answers. GUARANTEES

We research and specialize only in software for programmers of micros. If a product is unknown to you and we recommend it, then we take the risk.

PHILOSOPHY: We carry every programmer's product for CPM80, MSDOS and CPM86 plus every key product for APPLE, Commodore 64, ATARI and TRSDOS.

Ad Changed 12/30/83

"C" LANGUAGE

	LIST PRICE	OUR PRICE
APPLE: AZTEC C-Full, ASM	\$199	call
8080: BDS C-Fast, popular	150	125
8080: AZTEC C-Full	199	call
Z80: ECOSOF-Fast, Full	250	225
8086: C86- optimizer, Meg	399	call
8086 Lattice - New1.1 & 2.0	600	495
MicroSoft (Lattice)	MSDOS	500
Digital Research - Megabyte	8086	350
Desmet by CWare-Fast	8086	109

BASIC

	ENVIRONMENT	LIST PRICE	OUR PRICE
Active Trace-debug	8080/86	\$ 80	73
MBASIC-80 - MicroSoft	8080	375	\$255
BASCOM-86 - MicroSoft	8086	395	279
CB-86 - DRI	CPM86/PC	600	449
PBASIC - DRI	8086	150	119
Business BASIC-MicroS	MSDOS	600	449

EDITORS Programming

	ENVIRONMENT	LIST PRICE	OUR PRICE
BELLESOF - PASCAL	MSDOS	\$ 245	call
C Screen Editor - source	8080/86	NA	60
EDIX	PCDOS	195	149
FINAL WORD	8080/86	300	225
MINCE	CPM, MSDOS	175	149
PMATE - powerful	CPM	195	175
	8086	225	195
VEDIT - full, liked	CPM, PCDOS	150	119
	8086	200	159

SERVICES

- * Programmer's Referral List
- * Dealer's Inquire
- * Compare Products
- * Newsletter
- * Help find a Publisher
- * Rush Orders
- * Evaluation Literature
- * Bulletin Board

PASCAL

	ENVIRONMENT	LIST PRICE	OUR PRICE
PASCAL MT+86	CPM86	\$400	\$295
without SPP	CPM80	350	259
MS PASCAL 86	MSDOS	350	280
ALCOR PASCAL full w/	8080	250	call
right extensions to learn			
SBB PASCAL - great, fast	PCDOS	350	315
PASCAL 64 - nearly full	COM 64	99	89
UCSD PASCAL Compiler	UCSD-PC	375	call

LANGUAGE LIBRARIES

	ENVIRONMENT	LIST PRICE	OUR PRICE
C-to-dBASE - interface	8086/86	NA	\$95
C TOOLS - Graph. Str...	PCDOS	NA	115
FLOAT87 - Lattice. PL1	PCDOS	NA	115
GRAPH-GSX-80	CPM80	60	50
HALO	PCDOS	150	125
ISAM: Access Manager-86	8086	400	300
PHACT - with C	PCDOS	NA	250
FABS	CPM80	150	135
PASCAL TOOLS	PCDOS	NA	115
SCREEN: Display Mgr-86	8086	500	375
PANEL-86 - many	PCDOS	NA	315
WINDOWS - AMBER	PCDOS	295	call

FORTRAN

	ENVIRONMENT	LIST PRICE	OUR PRICE
MS FORTRAN-86 - Megabyte	MSDOS	\$350	\$265
SS FORTRAN-86	CPM-86	425	345
FORTRAN-80 - 66 decent	CPM80	500	350
INTEL FORTRAN-86	IBM PC	NA	1400

ASSEMBLER

	ENVIRONMENT	LIST PRICE	OUR PRICE
MAC/65 - macros, fast	ATARI	\$80	\$75
MACRO-80 -	CPM80	200	149
ORCA/M - Reloc. Macro	APPLE	149	129
RMAC, etc. (Prog. Util)	CPM80	200	159
RASM	8086	200	159

OPERATING SYSTEMS

	ENVIRONMENT	LIST PRICE	OUR PRICE
CPM-86	PC	\$60	\$49
"CPM86 Emulator"	MSDOS	NA	95
MicroShell upgrade CPM	8080	150	125
"MSDOS Emulator"	CPM86	NA	95
"Multitasking PCDOS"	PCDOS	239	179
QNX - real time UNIX	PC	650	call

TOOLKITS

	ENVIRONMENT	LIST PRICE	OUR PRICE
C Helper - DIFF, Flow	8080/86	NA	115
MicroTools	CPM80	150	125
NORTON for 1.1 or 2.0	PCDOS	80	65
"Multitasking Book with Disk"	PCDOS	80	65
Power	CPM80/PC	169	139
Programmer's Toolkit	8080/86	NA	95

COBOL

	ENVIRONMENT	LIST PRICE	OUR PRICE
CIS COBOL - no royalty	CPM80/86	\$800	\$600
Level II COBOL - High	8086	1600	call
MS COBOL-86	MSDOS	750	595
RM/COBOL-80	CPM80	750	595
RM/COBOL-86	8086	950	745

LOOKING FOR

*Fortran Level H with no changes
for PCDOS
*COMMERCIAL TOOLS - COMM 64

OTHER PRODUCTS

	ENVIRONMENT	LIST PRICE	OUR PRICE
ADA - Janus	CPM-80	\$300	\$259
ADA - Janus with tools	PCDOS	500	449
LISP - IQ LISP - full,	PCDOS	175	call
MAGIC/L - Intriguing	CPM80	295	call
MODULA II -	8086	495	455
MODULA II - PCode	APPLE	495	455
PL/1-86 - DRI	CPM-86	750	560
PL/1-80 - DRI	8080	500	369
ASCOM 80	CPM80	175	159
CACHE/Q - Virtual Mem.	CPM80/PC	225	185
MITE/BYE-8086/86	200	179	
"Read CPM86"	MSDOS	NA	95
LYNX - Overlay Linker	CPM80	250	215
PLINK-86 - Overlay, Meg	MSDOS	350	315
MSDOS Debugger	NA	135	
Trace-86		25	115

RECENT DISCOVERIES HIGHLIGHTED

C Helper includes source in C
for CP/M80, MSDOS for a:
DIFF, GREP Flowcharter, XREF,
C Beautifier and others. \$115.

Note: all prices subject to change without notice.

Call for a catalog, literature, comparisons, prices. Shipping \$2.50
per item purchased. All FORMATS available.

THE PROGRAMMER'S SHOP™

908-C Providence Highway, Dedham, MA 02026.
617-461-0120, Mass: 800-442-8070

VISA 800-421-8006 MASTER CARD

Circle no. 43 on reader service card.

16-BIT SOFTWARE TOOLBOX

by Ray Duncan

Bill Savage's floating-point benchmark program, discussed in this column in *DDJ*, September 1983, apparently caught the interest of a large number of readers. Versions in BASIC (9/83), PL/I (9/83), Fortran (1/84), Forth (11/83), Pascal (11/83), and C (Listing, page 96) have been contributed along with many timing reports.

In Table I (page 93) I have compiled the figures sent to me to date and sorted them in order of execution time (thank goodness for dBASE II). When duplicate timings on the same language and CPU were sent by different readers, I generally picked the more conservative results. A few languages were omitted altogether because of gross conflicts between two different reports (the IBM 4341 Fortran was one of these). I have also attempted to screen out any obvious typographical errors or blatant publicity grabs; nevertheless, let the reader beware!

The largest number of contributions came to *DDJ* via Gerald Edgar of Columbus, Ohio, who put the original benchmark program onto the CompuServe bulletin board system and asked for donations. This resulted in a set of over 50 timings, which he then sorted by degree of error, CPU, and language. After eliminating the duplications, I added these to my own table and designated them by the name "COMPUSERVE" in the "Source" column. Heartly thanks to all the anonymous CompuServe users who took the time to read and comment on this project.

Turning our attention to the table, the column "FPP" contains the type of hardware floating-point support that was used, if any was noted by the contributor. Most of the timings were accurate only to the nearest second (except for the very fast mainframes); they are displayed to three decimal places as an artifact of the dBASE II report program. The column "Error" was calculated as $ABS(2500.000 - A)$, converting to scientific notation and rounding the mantissa to the nearest integer. Obviously, the smallest errors correspond to the most accurate results.

This collection demonstrates very clearly that personal computers based on the Intel 8086 or 8088, when augmented with the 8087 floating-point coprocessor, can offer performance on this limited benchmark that is competitive with "super-minis" or mainframe computers costing ten to a hundred times more. Of

course, many significant issues such as the cost, size, and speed of mass storage, sophistication of systems tools, and ability to handle large numbers of interdependent processes are being totally ignored here.

We also received some advice and gentle chastisement from readers who are more knowledgeable on the subject of floating-point libraries and benchmarks than yours truly. I have excerpted a few of these letters below.

Jeffrey M. Speiser, of San Diego, California, writes: "It should be noted that for this benchmark the accuracy of the result is only suggestive of the accuracy of the floating-point arithmetic since:

- "1) Errors of opposite signs can cancel in the final result, suggesting greater than actual accuracy.
- "2) The errors can occur primarily over a small part of the argument range, suggesting much worse accuracy than is present over most of the argument range.
- "3) Careful examination of one case showed that not only was the error primarily in the TAN-ARCTAN pair, but it could be explained by assuming that both functions were computed exactly, with roundings only to the nearest machine-representable number. Moral: do a simple sensitivity analysis for any critical calculation, by using the first two terms in a Taylor series expansion."

Harry J. Smith, of Saratoga, California, writes: "... I duplicated [the previously published results] for Fortran-80, BASIC-80, and PL/I-80 with the standard floating-point libraries. But when I also displayed the results of each iteration, I made a startling discovery. Both the Fortran-80 and BASIC-80 programs reached their final value of 2304.863 and 2304.860, respectively, on the 2230th iteration and then repeated this same output for the next 270 iterations. If the original program had been asked to do 2305 iterations instead of 2500, then Fortran-80 and BASIC-80 would have looked quite good. The foregoing exemplifies the possibility of superficial tests leading to false conclusions..." He went on to describe in detail some procedures for evaluating

error in floating-point libraries, which we will save for a later column.

Karl J. Casper, Professor of Physics at Cleveland State University, contributed timings on machines ranging from the IBM 370 to the Hewlett-Packard 15C calculator. He commented: "This program... is somewhat unfair to the microcomputer if only single-precision numbers and functions are used. For example, I do not think that Microsoft really improved their algorithms from the Model I TRS-80 to the Color Computer. It is more probable that the increased accuracy results from increasing single precision to nine places on the CoCo.

"The problem is that the program begins to check angles very close to 90 degrees after only a few iterations. Most rational number approximations are designed to give good values for the functions over a wide range of arguments. It is certainly not clear that this program would correspond to any physically meaningful problems. Where physicists find that they are dealing with very small numbers, they would not usually try to evaluate these numbers by adding and subtracting large numbers. Physically meaningful benchmarks, however, would be very useful for evaluating computers.

"The point I am making is pretty simple. The Model I TRS-80 fails to get an accurate value simply because it lacks double-precision functions in BASIC. Both Pascal-80 and Fortran-80 (using double-precision functions) generate the correct value when run on the Model I, whereas Molinex 5.1 Pascal generates the same value as Level II BASIC. (It is, of course, comforting to find that both the Sinclair ZX-81 and the HP-15C calculator obtain accurate values.) For a scientist, speed is occasionally important, but the fact that some computers run at breakneck speed is hardly significant when the wrong answer is obtained. Checking this benchmark on the IBM 370 yielded even faster speeds than MicroFloat [hardware-assisted floating-point libraries]. But while an interactive Waterloo BASIC obtained the correct answer, IBM BASIC failed by a wider margin than any microcomputer."

I will continue to add results to the data base as they come in and will plan to publish the expanded collection in about six months. If any readers have proposals for a more fair, exhaustive, and physically

meaningful floating-point benchmark program, let's see those too! We have a noticeable deficiency of C language results compared to the huge number of compilers that are currently available. Some inter-

esting languages are missing altogether, including LISP, Logo, Ada, and Modula II. If you are kind enough to send in timing results, be sure to note whether single- or double-precision were used and the type

of hardware floating-point processor (if any). DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 196.

Table 1.
Floating-Point Benchmarks for DDJ

COMPUTER	MHZ	LANGUAGE	VERS	FPP	TIME ERROR (SEC)	SOURCE	COMMENTS
HONEYWELL		MULTICS FORTRAN			0.500 4E+1	SHERMAN GROMME	SINGLE PREC, TIME APPROX.
IBM 370		WATERLOO BASIC			0.570 2E-1	KARL CASPER	
DEC VAX 11/780		VMS FORTRAN-77			0.578 <1E-3	SHERMAN GROMME	SINGLE PREC
HP 1000-F		FORTRAN 4	2040		0.630 2E+2	COMPUSERVE	
IBM 3081		PL/I			0.660 2E-25	COMPUSERVE	
HONEYWELL		MULTICS FORTRAN			1.000 1E-3	SHERMAN GROMME	DOUBLE PREC, TIME APPROX
DEC VAX 11/780		VMS FORTRAN-77			1.054 <1E-3	SHERMAN GROMME	DOUBLE PRECISION
IBM 370		IBM BASIC			1.530 2E+3	KARL CASPER	PRESUMED SINGLE PREC
HP A700		FORTRAN 4			2.000 3E-8	COMPUSERVE	
IBM PC (8088)	4.77	WL SYSTEMS FORTH		8087	3.200 1E-3	JOHN GOTWALS	
8086	5.0	PL/I-86	1.01	8087	3.700 2E+1	BILL SAVAGE	MICROFLOAT LIBRARY
HP 9000 (68000)		FORTRAN (UNIX)			4.000 2E-6	COMPUSERVE	
HP 1000-F		FORTRAN FTN4	2040		4.000 5E-3	COMPUSERVE	
8086	5.0	PL/I-86	1.01	8087	4.000 2E+1	COMPUSERVE	
IBM PC (8088)	4.77	MVP-FORTH		8087	4.300 <1E-3	C. SPRINGER	
DEC PDP 11/44		RSX-11M FORTRAN		FIS	4.500 2E+2	JOHN TOSCANO	
DEC VAX 11/780		PASCAL	2.1		5.000 6E-10	COMPUSERVE	
IBM PC (8088)	4.77	MICROSOFT PASCAL	3.11	8087	6.000 <1E-3	J. SPEISER	DBL PREC, SEATTLE 87.LIB
IBM PC (8088)	4.77	MICROSOFT FORTRAN	3.1	8087	6.200 <1E-3	J. SPEISER	DBL PREC, SEATTLE 87.LIB
DEC PDP 11/44		RSX-11M FORTRAN		FIS	6.900 2E-1	JOHN TOSCANO	DOUBLE PREC
DEC PDP 11/34		RT-11 FORTRAN		FIS	7.500	J. SPEISER	
IBM PC (8088)	4.77	BASIC COMPILER	1.00	8087	7.800 1E-3	J. SPEISER	DBL PREC, SEATTLE 87.LIB
HP 1000F		BASIC			8.000 3E+2	COMPUSERVE	
8085	5.0	RMAC		8232	10.200 5E-3	BILL SAVAGE	MICROFLOAT LIBRARY
8085	5.0	PL/I-80	1.40	8232	10.400 5E-3	BILL SAVAGE	MICROFLOAT LIBRARY
8085	5.0	BASIC-80	5.20	8232	10.700 <1E-3	BILL SAVAGE	MICROFLOAT LIBRARY
8085	5.0	FORTRAN-80	3.40	8232	12.500 5E-3	BILL SAVAGE	MICROFLOAT LIBRARY
IBM PC (8088)	4.77	BASIC COMPILER	1.00	8087	14.200 <1E-3	J. SPEISER	DBL PREC, MICROWARE LIB
6502	4.0	UCSD P SYSTEM	II	8231	15.500 2E+2	STEVEN SPEARS	
8088	4.7	UCSD PASCAL	IV.1	8087	18.000 2E-7	COMPUSERVE	
APPLE II (6502)	1.75	MVP-FORTH		9511	18.500 2E+2	C. SPRINGER	
DEC PDP 11/44		RSX-11M BASIC-PLUS-2			21.800 1E+2	JOHN TOSCANO	SINGLE PREC
DEC MINC 11/23		FORTRAN		FPU	22.000 2E+2	BILL SAVAGE	
DEC PDP 11/44		RSX-11M BASIC			27.200 1E+2	JOHN TOSCANO	
HEATH H-89 (Z-80)	4.0	FORTRAN-80		9511	31.000 2E+2	JOHN TOSCANO	
DEC PDP 11/44		RSX-11M FORTRAN			34.600 1E+2	JOHN TOSCANO	
DEC MINC 11/23		MINC BASIC	2.0	FPU	38.000 2E+2	BILL SAVAGE	
HP 9186		HP BASIC			44.000 3E-7	COMPUSERVE	
HP 9836 (68000)		BASIC			44.290 <1E-3	BILL SAVAGE	
IBM PC (8088)	4.77	BASIC COMPILER	1.00		48.000 9E+1	J. SPEISER	SINGLE PREC
DEC LSI-11/23		C (UNIX)	V7		56.000 6E-8	COMPUSERVE	
DEC LSI-11/23		PASCAL-2	2.1A	FP	66.000 1E-7	COMPUSERVE	
WANG PC (8086)	8.0	BASIC INTERP	1.0.2		68.000 2E+2	RAY DUNCAN	

(Table continued on next page)

(Table I continued)

COMPUTER	MHZ	LANGUAGE	VERS	FPF	TIME ERROR (SEC)	SOURCE	COMMENTS
IBM PC (8088)	4.77	SUPERSOFT FORTRAN	1.07		70.000 9E+1	HUGH KAWABATA	SINGLE PREC
SUN WORKSTN (68000)		UNIX CC COMPILER			85.000 <1E-3	JOHN TOSCANO	
IBM PC (8088)	4.77	CI C86	1.33D	8087	85.000 1E-3	J. SPEISER	DOUBLE PREC
8086	5.0	BASIC-86	5.20		92.200 3E+2	BILL SAVAGE	
DEC PDP 11/44		RSX11M FORTRAN			103.000 2E-1	JOHN TOSCANO	DOUBLE PREC
DEC PDP 11/44		RSX-11M BASIC-PLUS-2			113.600 <1E-3	JOHN TOSCANO	DOUBLE PREC
68000	8.0	UCSD PASCAL	IV.12		115.000 3E-7	COMPUSERVE	
IBM PC (8088)	4.77	IBM APL			117.000 1E-8	J. SPEISER	
6809	2.0	PASCAL			119.000 8E+0	ANDY BALL	MICROWARE SYSTEMS CORP
HP 1000E		FORTRAN 4	2026		121.000 3E-8	COMPUSERVE	
8085	5.0	FORTRAN-80	3.40		140.800 2E+2	BILL SAVAGE	
8085	5.0	BASIC-80 COMPILER	5.20		140.800 2E+2	BILL SAVAGE	
HP 9835B		BASIC			140.800 1E-3	J. SPEISER	
ZENITH Z-100 (8088)		ZBASIC			142.730 2E+2	BILL SAVAGE	
6809	2.0	BASIC09			149.000 2E-2	ANDY BALL	MICROWARE SYSTEMS CORP
8088	8.0	SSS FORTRAN	1.04		154.000 2E-7	COMPUSERVE	
IBM PC (8088)	4.77	BASIC INTERPRETER			157.000 3E+2	J. SPEISER	SINGLE PREC
IBM PC (8088)	4.77	BASIC INTERPRETER	1.00		160.000 3E+2	J. SPEISER	DOUBLE PREC EXC TRANS.
IBM PC (8088)	4.77	BASIC COMPILER	1.00		170.000 1E-3	J. SPEISER	DOUBLE PREC
8086	8.0	MS PASCAL	?		171.000 1E-7	COMPUSERVE	
8085	5.0	BASIC-80	5.20		174.900 2E+2	BILL SAVAGE	
8086	5.0	PL/I-86	1.01		179.600 8E+2	BILL SAVAGE	
Z-80	4.0	BASIC-80	5.30		184.000 2E+2	COMPUSERVE	
HEATH H-89 (Z-80)	4.0	MBASIC COMPILER	5.2		189.700 2E+2	JOHN TOSCANO	
HEATH H-89 (Z-80)	4.0	FORTRAN-80			203.000 2E+2	JOHN TOSCANO	
TANDY 16B (68000)		XENIX CC COMPILER			211.000 1E-3	JOHN TOSCANO	
IBM PC (8088)	4.77	SUPERSOFT FORTRAN	1.07		211.000 <1E-3	HUGH KAWABATA	DOUBLE PREC
68000	6.0	C (TRS-XENIX)			212.000 1E-6	COMPUSERVE	
IBM PC (8088)	4.77	DRI PERSONAL BASIC	1.0		217.000 1E+3	RAY DUNCAN	SINGLE PREC
HEATH H-89 (Z-80)	4.0	MBASIC	4.8		229.800 2E+2	JOHN TOSCANO	
ZENITH Z-100 (Z-80)		MBASIC	5.22		236.740 2E+2	BILL SAVAGE	
8085	6.0	C/80	3.0		238.000 3E+2	COMPUSERVE	
DEC PDP 8		DS/8 FORTRAN IV			240.000 1E+3	SHERMAN GROMME	SINGLE PREC, TIME APPROX
8085	5.0	PL/I-80	1.30		254.400 8E+2	BILL SAVAGE	
IBM PC (8088)	4.77	DRI PERSONAL BASIC	1.0		255.000 1E+3	RAY DUNCAN	DOUBLE PREC
68000	6.0	BASIC/S-16	1.7		266.000 2E+2	COMPUSERVE	
HP-85 CALCULATOR		BASIC			280.767 1E-3	BILL SAVAGE	
HP-85 CALCULATOR		BASIC			281.000 6E-4	COMPUSERVE	
8088	8.0	COMP. INNOV. C86	1.33		288.000 7E-6	COMPUSERVE	
HP-75 CALCULATOR		BASIC			325.000 6E-4	COMPUSERVE	
Z-80	4.0	BASIC-E	2.2		330.000 5E+1	COMPUSERVE	
HEATH H-89 (Z-80)	4.0	C/80	3.0		370.700 3E+2	JOHN TOSCANO	
Z-80	4.0	PL/I-80	1.3		373.000 9E+2	COMPUSERVE	
OSBORNE I (Z-80)		C/80			420.000 2E+2	BOB BRIGGS	
TANDY I (Z-80)		FORTRAN-80			422.000 2E+2	KARL CASPER	SINGLE PREC
APPLE II (6502)	1.75	APPLESOFT			463.000 <1E-3	C. SPRINGER	
APPLE II (8088)	1.75	APPLESOFT BASIC			470.000 <1E-3	R. S. SCHLAIFER	
Z-80	4.0	NEVADA FORTRAN	2.2		473.000 8E+1	COMPUSERVE	
VIC-20 (6502)	1.0	BASIC	2.0		477.000 9E-5	COMPUSERVE	
TANDY I (Z-80)		MOLINERX PASCAL	5.1		485.000 1E+2	KARL CASPER	

(Continued on next page)

NEW! Back Issues

Act Now! This offer good only through March 31, 1984.

SAVE \$5.00 on this special back issue offer. Regularly \$3.50 per issue, now only \$2.50 each when you order a package of 5! For only \$12.50 you can have any 5 issues listed below.

#66 Volume VII, No. 4: 8080-Z80/8086 Cross-Assembler, Part 2 - Writing the Runic Compiler - Poor Person's Spelling Checker.

#68 Volume VII, No. 6: Multi-68000 Personal Computer - PDP-1802, Part One - Improved LET for LLL Basic - CP/M Print Utility.

#69 Volume VII, No. 7: IBM-PC Issue! CP/M-86 vs. MSDOS (A Technical Comparison) - Hi-Res Graphics on the IBM-PC - PDP-1802, Part II - Review of Word Processors for IBM.

#70 Volume VII, No. 8: Argum "C" Command Line Processor - SEND/RECEIVE File Transfer Utilities - Intel's 8087 - Performance Evaluation.

#71 Volume VII, No. 9: FORTH Issue! Floating-Point Package - H-19 Screen Editor - Relocating, Linking Loader - Z8000 Forth - Forth Programming Style - 8086 ASCII-Binary Conversion Routines - CP/M Conditional SUBMIT.

#72 Volume VII, No. 10: Portable Pidgin for Z80 - 68000 Cross Assembler, Part I - MODEM and RCP/Ms - Simplified 68000 Mnemonics - Nested Submits - 8086/88 Trig Lookup.

#73 Volume VII, No. 11: Wildcard UNIX Filenames - Tests for Pidgin - 68000 Cross Assembler Listing, Part 2 - Adding More BDOS Calls - The Perfect Hash - BASIC Memory Management - Benchmarks for CP/M-86 vs. MSDOS, and the 8087.

#75 Volume VIII, No. 1: Augusta, An ADA Subset for Micros - Xanadu: Hypertext from the Future - Stone Age Computers: 6000 Years of Computing Science - Small-C Compiler v2., Part 2.

#76 Volume VIII, Issue 2: PISTOL, A Forth-like Portably Implemented Stack Oriented Language - Program Linkage by Coroutines, Forth to BASIC - Linking CP/M Functions to Your High-Level Program - Concurrent CP/M-86 - Small Systems Engineering CP/M-80 Expansion Card for the Victor 9000 - REVAS Disassembler.

#77 Volume VIII, Issue 3: Augusta, Part II: The Augusta P-Code Interpreter - A Small-C Operating System - 6809 Threaded Code: Parametrization and Transfer of Control - A Common-Sense Guide to Faster, Small BASIC - A Fundamental Mistake in Compiler Design - Basic Disk I/O, Part I.

#78 Volume VIII, Issue 4: RECLAIM Destroyed Directories - Binary Magic Numbers - 8080 Fig-Forth Directory & File System - SAY" Forth Votrax Driver - TRS-80 8080 to Z80 Translator - Basic Disk I/O, Part II.

#79 Volume VIII, No. 5: Augusta Part III: The Augusta Compiler - A Fast Circle Routine - Enhancing the C Screen Editor - Shifts and Rotations on the Z80 - The SCB, TSX, and TXS Instructions of the 6502 and 6800 - MS-DOS vs. CP/M-86 - Controlling MBASIC - The Buffered Keyboard - IBM PC Character Set Linker - Flip Utility for the IBM PC.

#80 Volume VIII, Issue 6: Fast Divisibility Algorithms - B-Tree ISAM Concepts - CP/M BDOS and BIOS Calls for C - Serial Expansion in Forth - Fast Matrix Operations in Forth, Part I - Yes, You Can Trace Through BDOS - Julian Dates for Microcomputers - 8088 Addressing Modes - 8088 Line Generator - CP/M Plus.

#81 Volume VIII, Issue 7: Augusta, Part IV (The Augusta Compiler, continued) - RED: A Better Screen Editor, Part I - Anatomy of a Digital Vector and Curve Generator - Fast Matrix Operations in Forth, Part II - The AGGHHH! Program - MBOOT Revisited - CP/M Plus Feedback - MS-DOS Rebuttal - 68000 Tools - Sizing Memory on the IBM PC.

#82 Volume VIII, Issue 8: Serial-to-Parallel: A Flexible Utility Box - McWORDER: A Tiny Text Editor - And Still More Fifth Generation Computers! - Specialist Symbols and I/O Benchmarks for CP/M Plus - CP/M Plus Memory Management - Zero Length File Test - PAUSEIF, QUITIF, and now SKIPIF - ACTxx Cross Assemblers.

#83 Volume VIII, No. 9: FORTH ISSUE! Forth and the Motorola 68000 - Non-deterministic Control Words in Forth - A 68000 Forth Assembler - GO in Forth - Precompiled Forth Modules - Signed Integer Division - Some Forth Coding Standards - The Forth Sort - A speed and accuracy benchmark program for high-level languages - Using a Digital Spreadsheet Program for Something Fun and Unusual.

#84 Volume VIII, No. 10: DDJ's new C/Unix column! - Unix to CP/M Floppy Disk File Conversion - A Small-C Help Facility - Attaching a Winchester Hard Disk to the S-100 Bus - Using Epson Bit-Plot Graphics - Your Fantasy Computer System - 8086/88 Function Macros - Auto Disk Format Selection - CP/M Plus Device Tables.

#85 Volume VIII, Issue 11: A Kernel for the MC68000 - A DML Parser - Towards a More Writable Forth Syntax - Simple Graphics for Printer - 8080 to Z80 Program Conversion, CP/M Plus DPB Macro Fix, and Quicker Submit File Truncation - Floating-Point Benchmarks and an MSDOS COM File Loader - software and book reviews.

#86 Volume VIII, Issue 12: Faster Circles for Apples - Cursor Control for Dumb Terminals - Dysan's Digital Diagnostic Diskette - Building a Programmable Frequency Synthesizer - Unix and Non-Interactive, User-Unfriendly Software - Interfacing a Hard Disk Within a CP/M Environment - The New MS-DOS EXEC Function.

#87 Volume IX, Issue 1: NBASIC: A Structured Preprocessor for MBASIC - A Simple Window Package - Forth to PC-DOS Interface - Sorted Diskette Directory Listing for the IBM PC - Emulate WordStar on TOPS-20 - More on optimising compilers - Problems under CP/M Plus with PAUSE - The PIP mystery device contest - Microsoft BASIC - An improved CLINK utility.

#88 Volume IX, Issue 2: Telecommunications Issue! Micro to Mainframe Connection - Communications Protocols - Unix to Unix Network Utilities - VPC: A Virtual Personal Computer for Networks - PABX and the Personal Computer - BASIC Language Telecommunications Programming - U.S. Robotics S-100 Card Modem - Sysdrive program, bringing up CP/M Plus - PCDOS Close Function - The Microsoft Assembler - more on C layout standards - book and software reviews.

TO ORDER: Send \$3.50 per issue (or \$12.50 for every set of 5) to: Dr. Dobb's Journal, P.O. Box E, Menlo Park, CA 94026.

Please send me the issue(s) circled: 66 68 69 70 71

Name _____

72 73 75 76 77 78 79 80

Address _____

81 82 83 84 85 86 87 88

City _____

I enclose \$ _____ (U.S. check or money order).

State _____ Zip _____

Outside the U.S., add \$.50 per issue.

Please charge my: ☐ Visa ☐ M/C ☐ Amer. Exp.

Card No. _____

Exp. Date _____

Signature _____

Availability on first come/first serve basis. Outside the U.S., add \$.50 per issue ordered. Price includes issue, handling, and shipment by second class or foreign surface mail. Within the U.S., please allow 6-9 weeks to process your order second class. For faster service within the U.S., we'll ship UPS if you add \$1.00 for 1-2 issues and \$.50 for each issue thereafter. We need a street address, not a P.O. Box. Outside the U.S., add \$1.50 per issue requested for airmail shipment.

(Table I continued)

COMPUTER	MHZ	LANGUAGE	VERS	FPP	TIME ERROR (SEC)	SOURCE	COMMENTS
TANDY I (Z-80)		BASIC			512.000 1E+2	KARL CASPER	
COMMODORE 64 (6510)		BASIC			514.000 <1E-3	TERRY THOMAS	
Z-80	4.0	ZBAS	1.4		540.000 2E+2	COMPUSERVE	
TANDY CC (6809)		FORTH			560.000 <1E-3	GARY BERGSTROM	
TANDY CC (6809)		BASIC RS-CC	1.0		585.000 <1E-3	GARY BERGSTROM	
ZENITH ZW110 8088	5.0	CI C86			655.000 <1E-3	A. F. HERBST	
IBM PC (8088)	4.77	CI C86	1.33D		695.000 1E-3	J. SPEISER	DOUBLE PREC
TANDY 16B (68000)		XENIX MBASIC			773.600 1E-3	JOHN TOSCANO	
IBM PC (8088)	4.77	BASIC INTERPRETER	2.0		890.000 1E-3	J. SPEISER	DOUBLE PREC
Z-80	2.0	NEVADA FORTRAN	3.0		975.000 8E+1	SHERMAN GROMME	
8085	6.0	CB-80	1.2		997.000 2E+1	COMPUSERVE	
8085	6.0	FORTTRAN-80	3.4		1251.000 1E-12	COMPUSERVE	
8085	6.0	CBASIC	2.06		1623.000 2E+1	COMPUSERVE	
Z-80	4.0	BASIC-80	5.3		1980.000 1E-7	COMPUSERVE	
Z-80	4.0	AZTEC C II	1.05		2190.000 1E-5	COMPUSERVE	
HEATH H-89 (Z-80)	4.0	AZTEC C	1.05		2226.000 <1E-3	JOHN TOSCANO	
TI-59 CALCULATOR					3240.000 7E-3	COMPUSERVE	
Z-80	2.5	S-BASIC			3900.000 3E+4	COMPUSERVE	
TANDY I (Z-80)		FORTTRAN-80			4602.000 <1E-3	KARL CASPER	DOUBLE PREC
TANDY I (Z-80)		PASCAL-80			5220.000 <1E-3	KARL CASPER	
Z-80	4.0	JRT PASCAL	2.1		5760.000 1E-3	COMPUSERVE	
HP-65 CALCULATOR					6000.000 9E-1	COMPUSERVE	
HP 15C CALCULATOR					10200.000 1E-2	KARL CASPER	
SINCLAIR ZX-81		BASIC			86400.000 3E-1	KARL CASPER	ABOUT 1 DAY

End Table I

16-Bit Toolbox Listing (Text begins on page 92)

```

main()
/* Floating point accuracy test adapted from DDJ 9/83 p. 122 */
{
    int i, iloop=2500;
    double a=1;
    double tan(), atan(), exp(), log(), sqrt();

    for (i=1; i<iloop; i++)
    {
        a=tan(atan(exp(log(sqrt(a*a)))))+1.0;
    }
    printf("\na = %10.4f\n", a);
}

```

End Listing

BDS C

The *fastest* CP/M-80 C compiler available today

Version 1.5 contains some nifty improvements:

The unscrambled, *comprehensive* new User's Guide comes complete with tutorials, hints, error message explanations and an index.

The CDB symbolic debugger is a valuable new tool, written in C and included in *source form*. Debug with it, and *learn* from it.

Hard disk users: You can finally organize your file directories sensibly. During compilation, take advantage of the new path searching ability for all compiler/linker system files. And at run-time, the enhanced file I/O mechanism recognizes user numbers as part of simple filenames, so you can manipulate files located *anywhere* on your system.

BDS C's powerful original features include dynamic overlays, full library and run-time package source code (to allow customized run-time environments, such as for execution in ROM), plenty of both utilitarian and recreational sample programs, *and speed*. BDS C takes less time to compile and link programs than any other C compiler around. And the execution speed of that compiled code is typically lightning fast, as the Sieve of Eratosthenes benchmark illustrates. (See the January 1983 BYTE, pg. 303).

BD Software 8" SSD format, \$150
P.O. Box 9 Free shipping on pre-paid orders
Brighton, MA 02135 Call or write for availability on
(617) 782-0836 other disk formats

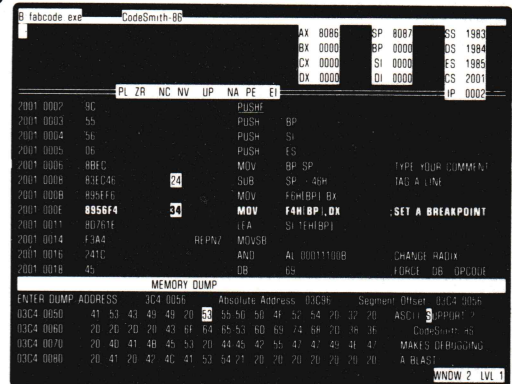
Circle no. 5 on reader service card.

NEW
IBM-PC
DEBUGGER

"...by far the best debugging tool available for the IBM-PC today."

S. Lerner, Multi Systems Group

CodeSmith™-86



- Simple/single keystroke commands
- Scroll up/down thru full-screen disassemblies
- SCREENSAVE Mode
Saves and restores user's graphic display when breakpoint hit
- Continuous monitoring of selected memory
- Hundreds of tagged breakpoints supported
- Disassemble selected ranges of memory code to disk—compatible w/IBM Assembler
- Many other inspired features

VISUAL AGE Requires MS-DOS & 160K RAM (min.)
MasterCard • VISA • COD Orders
642 N. Larchmont Blvd., Los Angeles, CA 90004
(213) 439-2414 CodeSmith, TM International Arrangements, Inc.
MS, TM Microsoft Corp. IBM, TM International Business Machines Corp.

INTRODUCTORY OFFER ONLY
Call residents add 6.5% sales tax
\$145.00

Circle no. 66 on reader service card.

DeSmet C \$109

PCDOS+ — CP/M-86++ — MP/M-86++ — CCP/M-86++

• C DEVELOPMENT PACKAGE

C Compiler, Assembler, Linker, and Librarian
Full Screen Editor — Native 2.0 Support

• COMPLETE IMPLEMENTATION

Everything in **K & R** (incl. **STDIO**)
Small case — 64K code, 64K data/stack
All 8088/8087 data types
Intel assembler mnemonics

• OUTSTANDING PRICE/PERFORMANCE

Aug. '83 **BYTE** "SIEVE" Benchmark
135 bytes compiled — 6144 bytes linked
65 sec. compile (fdisk) — 11.5 sec. run (10 iter)

• SUPPORT LIBRARIES

Both 8087 and Software Floating Point
PC Keyboard/Display
Trig/Log functions

• UPDATES

Updates and Bugs reported in newsletter
No time/number limits

+ Trademark IBM ++ Trademark Digital Research

ORDER FORM

Name _____

Address _____

Phone _____

OS: ☐ PCDOS ☐ MSDOS ☐ CP/M-86, MP/M-86, CCP/M-86
DISKS: ☐ 8" (3740) ☐ 5.25" SS (160KB) ☐ 5.25" DS (320KB)

MACHINE? _____ (IBM, ZENITH, NEC, etc.)

DeSmet C Package \$109

Manual Only (refunded with order) \$20

Update Only \$20

Sub Total _____

California Residents add 6% Sales Tax _____

Send To: **C WARE**

P.O. Box 710097

San Jose, CA 95171-0097

(408) 736-6905

Total _____

Circle no. 16 on reader service card.

by Michael Wiesenber

Faster than a Speeding 8088

Lightning 286, from Lomas Data Products, is an S100 80286 board that runs 8088 and 8086 software four times faster than a 5 MHz 8086 (which itself is considerably faster than the 8088), and can run the 80287 high-speed numeric processor, giving program execution *in some programs* a doubling of speed. The Lightning 286 can run at clock rates up to 10 MHz, and Lomas suggests that they will increase their rates as Intel improves the 80286 (which currently runs at only 6 MHz). Lightning 286 allows 16 Mb physical address space and 1 Gb virtual address space. Four operating systems, MS-DOS, CP/M-86, Concurrent CP/M-86, and MP/M-86, are currently available with the board, which conforms to IEEE 696 S100 bus specifications. The board costs \$1395, with optional 80287 for another \$550. **Reader Service No. 101.**

High Speed Modems

Prentice Corporation's limited distance, **ALD/1**, full and half duplex, asynchronous 1200 to 9600 baud modem conforms to Bell specifications 43401 and 41028 with metallic, twisted-pair Local Area Data Service (LADS) for local office and campus networks. It comes in a stand-alone version for \$300 and rack-mountable for \$200. The **HSLD/1**, 2400 to 19,200 baud, is \$490 and \$390. The synchronous leased line 4800 baud **208A/B** is \$1750 and \$1650, while the 9600 baud **9629** for four-wire 3002-type unconditioned leased lines in point-to-point applications is \$2750 and \$2650. **Reader Service No. 107.**

Here Come the Little Guys

Even though I rarely mention products for which the press release lists no price, I'd like to describe this one because it shows what you can shortly expect many computers to look like. The **UDI-500**, from Universal Data Incorporated, is the first portable computer with internal batteries and *two* disk drives. At 11 by

13 by 3 1/8 inches and 12.8 pounds, it is less than half the size and weight of competitors. The CMOS Z80 permits up to 12 hours of operation on one charge of the nickel-cadmium batteries. You can get 80 hours if you never use the disks or two with constant disk use. The batteries charge fully in 10 hours. The LCD has eight lines of 40 characters. An accessory slot accommodates the optional 300 or 1200 baud direct-connect modem. The low-power static CMOS memory is 64K standard, expandable to 256K. Two plug-in CPU boards (both CMOS) come with the computer, Z80 and 1805. You get CP/M 2.2 and Micro-DOS. Available software includes text processor, spelling checker, data filing system, spread sheet, communications package, and various flavors of BASIC. **Reader Service No. 103.**

Computers Talking to Each Other

SOFTCOM Telecommunications Utility, from the Software Store, is an intelligent terminal program and CP/M to CP/M file transfer utility that supports XON/XOFF and transmits at up to 9600 baud in full or half duplex. It runs on 8080, 8085, and Z80 systems with at least 32k and CP/M, and costs \$150. **Reader Service No. 109.**

CP/M Plus for Cromemco

Super BIOS Plus, from MICAH (Micro Applications and Hardware), is CP/M Plus for Cromemco computers, starting at \$495. You get source code for all modules. You can also add the **Expand** program, a CDOS emulator, and the Select Word Processor. MICAH also carries Morrow hardware configured for Cromemco. **Reader Service No. 115.**

Don't Give Me Any Static

Staticide Wipes, from ACL Incorporated, are individually packaged towelettes for cleaning and static control of video screens, packed in square

foil packs, 24 to a box, \$4.98 per box. ACL offers free samples and information on other Staticide products. **Reader Service No. 119.**

Come Forth

The Institute for Applied Forth Research is hosting the **1984 Rochester Forth Applications Conference** June 5 to 9 at the University of Rochester. Speakers will discuss real-time systems. The conference sponsors are calling for papers on Forth applications and techniques. A 200-word abstract is due April 1 and the paper should be in by May 1. Papers should be on real-time software (including process control, data acquisition, smart instrumentation, laboratory systems, robotics, computer vision, spacecraft navigation, music and voice synthesis), applications (in particular, Forth microchip applications), or Forth technology (finite state machines, control and data structures, and hybrid hardware and software systems). Papers should be sent to Diane Ranocchia (who can be contacted for more information), Institute for Applied Forth Research, Inc., 70 Elmwood Avenue, Rochester, New York 14611; (716) 235-0168.

Proportional Spacing on Wordstar

Proportional Spacing on WordStar, from the Writing Consultants, is a \$20 book that tells you how to do what (they claim) "the industry has said WordStar needs," plus print two or more justified columns on a page and underline `_spaces_ between _words_`. **Reader Service No. 117.**

UNIX for the People

Will UNIX be on microcomputers? **Real UNIX? The Office UNIX System** from Unisource is not a lookalike, but full UNIX, licensed from AT&T, and developed as VENIX/86 by VenturCOM for the IBM PC. Unisource's Of-

MEGA-BYTE

SPECIALIZING IN HARD DISKS AND HIGH PERFORMANCE STREAMING TAPE UNITS

MD - 10 11 MEGABYTE (FORMATTED) 5 1/4" HARD DISK Including **POWER!** \$1995

POWER!

Check out InfoWorld's Rave review of **POWER** Nov. 8/82)

55 CP/M UTILITIES IN ONE PACKAGE

...The super program that puts you in control of CP/M (or MS-DOS) and your winchester.

POWER automatically numbers disk files. Just pick file number to Copy, Erase, Reclaim, Type, etc.

...Your computer feeds the file names automatically. You do NO typing! NO typing error ever!

YOU DON'T NEED SYSTEM DISK IN ANY DRIVE.

No more BDOS ERROR!

YOU Test and Fix bad disks! Reclaim accidentally erased files or programs! Single step thru memory up or down! Search, view, change memory or disk in a snap! See Status and File Size instantly! Verify check-sums for programs! Load or Save programs at any address.

55 prompted user-friendly functions for housekeeping and a 120 page easy-read user's guide make POWER your most often used software. You'll use it every day!

Now, POWER! also includes a special program that lets you lock sensitive files, so that only you can access them. Without the secret PASSWORD which you can create and change at will, no prying eyes will ever know your secret file even exists.

SOME MAJOR POWER COMMANDS:

REN	CM	WRITEGR	DISK	STAT
TYPEA	USR 2	DUMPA	SPEED	SETDIR
SIZE	ERA	SEARCH	WRITE	RECLAIM
XUSER	TYPEX	?	DUMPIX	DS
SETRO	CHECK	COPY	FILL	READ
GROUP	TEST	TYPEH	EX	DUMP
SAVE	SETWR	EXIT	DIR	MOVE
READGR	LOG	USER	TYPE	JP
DUMPH	LOAD	SETSYS	RUN	

For this low price your winchester will be delivered completely assembled and tested, with drive controller, case, power supply, cabling, Z-80 interface and the best disk controller software on the market.

- The unique and simple interfacing system does not tie up existing ports.
- Significantly faster than other winchester subsystems which interface through the IEEE-488 port.
- CP/M drivers require minimum memory overhead (about 2K-other systems require as much as 6k)
- The MD-10 can read or write a 64K file in less than four (4) seconds.
- A network system will be available shortly which can support up to sixteen mixed types of computers from one MD-10 or larger disk subsystem.
- With POWER! Software files can be code word protected.
- Other systems available: 22 and 44 megabytes

MD-20 with 22MB formatted: \$2895

MD-44 with 44MB formatted: \$3995

- Up to eight (8) winchester subsystems can be interfaced to one computer.
- Software supports 32 different user areas per MD-10.
- Backing up hard disk files is simple with the special software which is provided with all subsystems.
- With a single hard disk installation, the MD-10 subsystems becomes units A and B with the standard drives being designed E and F if a second MD-10 is installed later it becomes units C and D.
- MD-10 or larger systems will interface with IBM PC or any Z-80 computer (CCS, APPLE (CP/M), ZENITH/HEATH, NORTHSTAR, GODBOUT, XEROX 820, Z-80/S100, ALSPA, or TRS-80 MOD II) using CP/M, OASIS, PCDOS 2.0, and Godbout Software supports both CP/M and MP/M816.
- Double density modifications are available allowing you to later increase the capability of an MD-10 to about 20 megabytes, MD-20 to 40, etc.
- **Full One-Year Warranty**

HIGH PERFORMANCE STREAMING TAPE

Unique! Simple! Compatible!

- Field-proven Archive Sidewinder tape unit.
- Interfaces with ANY SASI/SCSI compatible hard disk peripheral.
- Full CRC error-checking
- Superb reliability.
- Full One-Year Warranty.
- Fast (up to 20MB in less than 5 minutes, up to 40MB in less than 10 minutes).

\$1795.00 20MB

\$1995.00 45/60MB

\$29.95 3M DC300XL

Streaming Tape Cartridges

MEGA-BYTE

ORDER Name	Qty.	Price	Total
VISA <input type="checkbox"/> M/C <input type="checkbox"/> COD <input type="checkbox"/>		Total Order	
Card #	Exp. Date	If Utah Tax	
Signature	Total Enclosed		



TO ORDER OR FOR MORE INFORMATION:
(801) 257-7033 or mail to MEGA-BYTE INC. RT. 3 Box 35, Tremonton, Utah 84337
TERMS: Cashiers Check, VISA, M/C or COD.
Shipping charges added to all orders.



NAME
COMPANY TITLE
ADDRESS CITY
STATE ZIP
TEL ()

CP/M is a registered trademark of Digital Research. OASIS is a proprietary product of Phase One Systems Inc. Z-80 is a trademark of Zilog. MD-10 is a registered trademark of Media Distributing.

Office Menu Tool uses prepared menus to run applications or execute commands, and you can make your own menus. You get four editors, C compiler, BASIC, assembler, uucp, cu, nroff, a spelling checker, table formatter, library routines, vi, more, termcap, and the c shell, in single user format for \$800 and multiuser for \$1000. The Office Menu Tool is \$125. You can add the FinalWord for \$395, Leverage DBMS for \$385, ViewComp spreadsheet for \$395. There are also various single and multiuser combinations, with VENIX/86 plus one or more ap-

plications, starting at \$1125. **Reader Service No. 111.**

Fortran 77

Those of you who use Fortran 66 or, heaven forbid, Fortran 2 and are tired of the gibes from your friends who use Pascal ("Nyah, nyah, you don't have good string-handling capabilities." "Your language ain't structured.") will be thrilled to learn that Absoft has **FORTAN 77** for the Whitesmiths Idris operating system

on 68000 systems. This is an ANSI Fortran 77 compiler that generates position-independent and reentrant object code, supports virtual arrays, has a full screen source level symbolic debugger, and enables dynamic linking of modules at run time. It compiles up to 3500 lines of code per minute and has no restrictions on code size. You'll pay \$695 to \$2000. **Reader Service No. 113.**

Reader Ballot

Vote for your favorite feature/article. Circle Reader Service No. 197.



And a Small Powerhouse

How about a portable computer with a 132-column screen? The **Execuport XL** from Computer Transceiver Systems Inc. is just that, with Z80 and CP/M, while the **Execuport XL+** has both the Z80 and 80186 (true 16-bit) and MS-DOS. Both come bundled with Perfect Calc, Speller,

Writer, and Filer. Optionally, you can get OASIS, CP/NOS, CP/NET, and MP/M. The XL has 80K, while the XL+ has 128K, which can be expanded. Both have a 9-by-5 green phosphor screen with 132 columns by 24 lines, two double-sized, double density 5¼-inch drives with 800K storage

each, and 22 user-programmable function keys. The XL is \$2695 and the XL+ \$3495. You can lease them for, respectively, \$98 and \$130 per month. The computers are supported by CTSI's nationwide on-site sales and service programs. **Reader Service No. 105.**

DDJ

Q: What's easier than spending money?

A: Keeping track of it with... **Checks & Balances**

At last! A *full screen* editing checking and finance program so informative and easy to use you'll wonder how you lived without it! *Especially at \$49⁹⁵*

Sixteen user-definable categories.

Category totals instantly updated with each new entry or change.

Screen always shows four entries. Use your cursor to add or change any entry anytime. Scroll forward and back through your register or instantly go to any entry.

Includes categories for bills, cash expenditures, deposits and service charges.

Checks and Balances aligns decimals and left justifies text.

For complete records, cash expenditures are included in category totals but not bank balance.

Entries which have cleared through your bank.

All commands in plain English.

True checking balance instantly updates with each entry.

Bank statement balance reflects only items checked off as having cleared your bank.

The screenshot displays a terminal window with the following content:

```

4289.90 A  =ADVERTISING
253.67 LABR =CONTRACT LABOR
4139.23 DEP =DEPOSITS
132.51 ENTR =ENTERTAINMENT
2954.86 ER  =EQUIPMENT FOR RESALE
3440.12 E   =EQUIPMENT-IN HOUSE
1370.88 DEPM =MERCHANT DEPOSITS
480.60 MISC =MISC. EXPENSES
0.00 BILL   132.51 CASH
-----
# 1317 06/14/83 Pay TO: MARVIN JONES $ 55.00 X
      <A> Memo: PRINTING & LABOR
# 1318 06/10/83 Pay TO: ALLENBACH INDUSTRIES $ 267.20 X
      <ER> Memo: INVOICE #7830, 7904
#CASH 06/10/83 Pay TO: TEMPURA INN $ 132.51
      <ENTR> Memo: MEETING TO DISCUSS NEW ADVERTISING
#DEP 05/21/83 Pay TO: DEPOSIT $ 240.00 X
      <DEP> Memo: SALES FROM MAGAZINE ADD
-----
File=CDE = DISPLAY & EDIT = TRUE $ 12209.42 BANK $ 7750.41
SHOW
  
```

So easy to use! See your financial totals start to add up before your eyes in less than five minutes! Checks and Balances' cursor can move anywhere, anytime, to add or change entries, and all totals will instantly be updated on your screen when you change an amount or move a check to a different category. Checks and Balances gives you complete printed reports of the entire register, each category, or reports based on any entry in "Pay to" or "Memo"—even bills and uncashed or missing checks. File size is limited only by your disk capacity. Over 32,000 entries are possible.

Technical: 80 x 24 CRT with addressable cursor, reverse video optional. Z-80 processor, one or more disk drives with over 180k capacity. 56k RAM.



CDE SOFTWARE
2463 McCready Avenue
Los Angeles, CA 90039



ADVERTISER INDEX

Reader Service No.	Advertiser	Page No.	Reader Service No.	Advertiser	Page No.	Reader Service No.	Advertiser	Page No.
1	Adler Computer Systems Co.	81	21	Ecosoft, Inc.	14	47	Quest Research	53
2	Ashton-Tate	2	22	Foehn Consulting Engineering.	77	48	Quic-n-Easi Products, Inc..	13
3	Avocet Systems, Inc.	11	23	GGM Systems	67	49	Rational Systems	102
4	Awarco Active Software	74	24	GTEK, Inc.	81	*	Edward Ream	68
5	BD Software.	97	25	Hallock Systems Consultants	83	51	Revasco	64
6	Bonnie Blue Software	29	73	Harvard Softworks	58	*	Rocky Mountain Software Systems.	17
*	Borland International.	104	26	Hawkeye Grafix.	63	52	Sage Computer Technology	23
8	California Digital Engineering	101	27	Laboratory Microsystems	3	53	Seattle Computer	7
9	The Code Works.	91	28	Lifeboat Associates.	61	54	SemiDisk Systems.	71
10	Computer Design Labs	55	29	Logical Systems	27	55	Shaw Laboratories	20
11	Computer Friends.	46	30	Loki Engineering	65	56	Simpliway Products Co.	57
12	Computer Innovations	45	31	Manx Software	4	57	SLR Systems	39
13	Computer Resources of Waimea.	53	33	Mega-Byte	99	58	The Software Store	56
14	Creative Solutions.	49	34	MicroMotion.	60	59	Software Writers Intn'l Guild	59
15	C Users' Group	64	35	Micro Processors Unlimited	57	60	Software Writers Intn'l Guild	33
16	C Ware	97	36	Modal Systems	4	61	Solution Technology, Inc.	21
*	DDJ Advertising.	56	37	Mylstar Electronics, Inc.	41	62	Solution Technology, Inc.	19
*	DDJ Change of Address	102	39	Novum Organum	63	63	Syntax Constructs, Inc.	56
71	DDJ Subscribe.	69	40	Overbeek Enterprises	77	64	Telecon Systems	31
*	DDJ Back Issues.	95	*	John D. Owens Associates, Inc.	82	65	Unified Software Systems.	67
*	DDJ Reston	87	41	Pacifica Technology	89	32	Video Games International	15
72	DDJ Bound Volume	85	42	Pascal & Associates	51	66	Visual Age	97
17	Data Access Corporation.	8	43	The Programmer's Shop	91	67	Whitesmith, Ltd.	103
18	Data Management Associates, Inc.	25	44	PRO Microsystems	91	68	Mark Williams & Co.	35
19	Dedicated Micro Systems, Inc.	82	45	PRO Microsystems	57	69	Workman & Associates	27
20	D & W Digital	43	46	Puterparts	81	70	Writing Consultants.	31

C Programmers: Program three times faster with Instant-C™

Instant-C™ makes programming three or more times faster by eliminating the time wasted by traditional compilers. Many repetitive programming tasks are automated to make programming less frustrating and tedious.

- Two seconds elapsed time from completion of editing to execution.
- Full-screen editor integrated with compiler; compile errors set cursor to trouble spot.
- Editor available any time during session.
- Symbolic debugging; single step by statement.
- Automatic recompilation when needed. Never a mismatch between source and object code.
- Directly generates .COM, .EXE, or .CMD files.
- Follows K & R—works with existing source.
- Single, integrated package.
- Works under PC-DOS*, MS-DOS*, CP/M-86*.

More productivity, less frustration, better programs. **Instant-C™** is \$500. Call or write for more information.

Rational
Systems, Inc.

(617) 653-6194
P.O. Box 506
Natick, Mass. 01760

*[Trademarks: PC-DOS (IBM), MS-DOS (Microsoft), CP/M-86 (Digital Research, Inc.)
Instant-C (Rational/Systems, Inc.)]

Changing Your Address?

Staple your label here.

To change your address, attach your address label from the cover of the magazine to this coupon and indicate your new address below.

Name _____

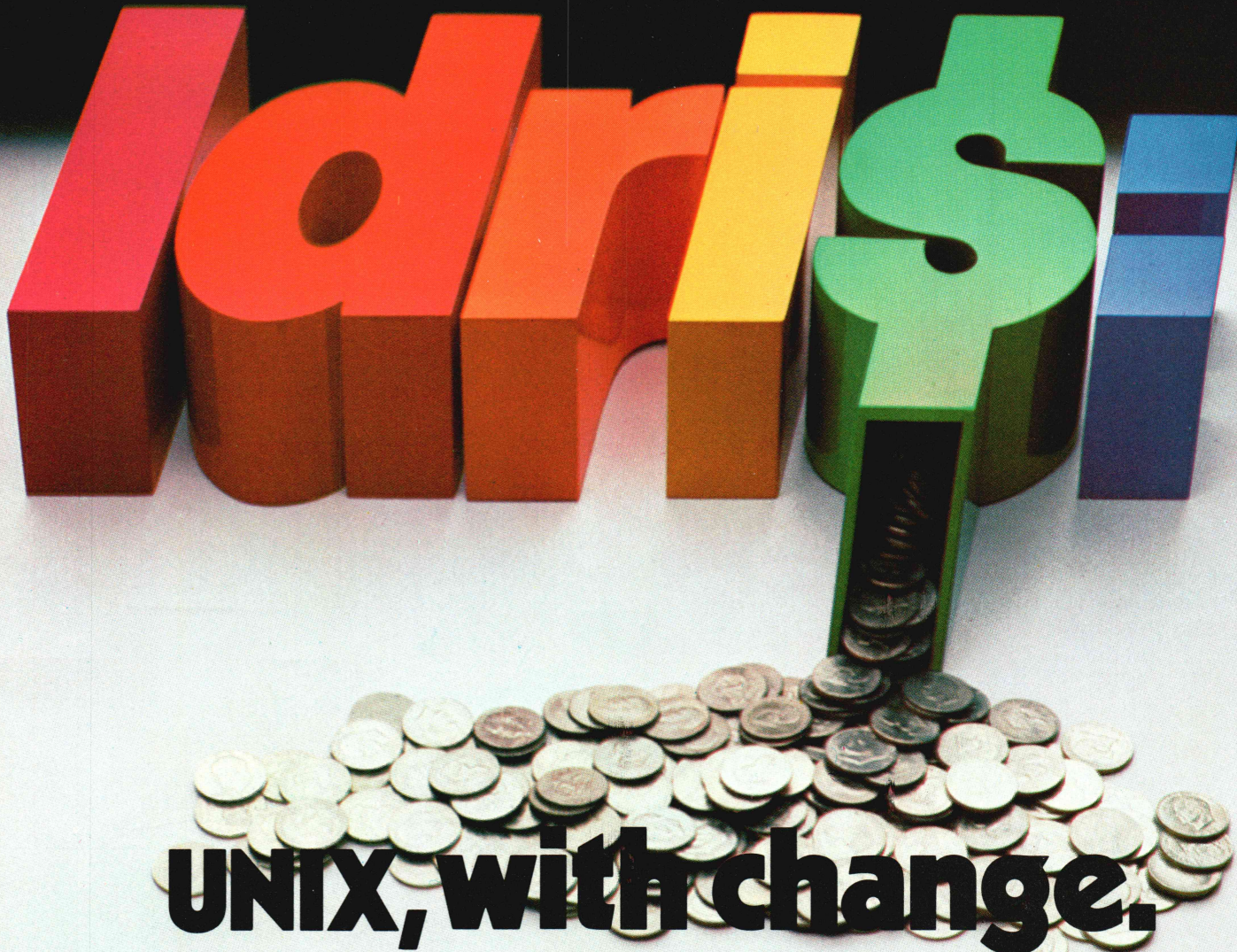
Address _____

Address _____ Apt. # _____

City _____ State _____

Zip _____

Mail to: DDJ, P.O. Box E, Menlo Park, CA 94026



With Idris, developers get the functionality, compatibility and portability of UNIX. And pocket the Idris difference.

• **GREATER PORTABILITY.**

Applications developed under Idris on any micro compiler can run on any other which supports Idris. These are PDP-11s (including PRO-350 and Micro-11), numerous Motorola 68Ks and the 8086/88 based IBM PC and DEC Rainbow.

Idris runs under MS/DOS as an application!

• **COMPLYING WITH THE UNIX USER GROUP STANDARDS.**

• **TWICE THE NUMBER OF USERS as UNIX** on comparable hardware, because Idris is optimized for microprocessors.

• **MORE DISK SPACE FOR FILES AND PROGRAMS.** Idris occupies less than 1.5 megabytes of disk.



• **MORE COST EFFECTIVE PER USER.**

The \$550 end-user price is for as many users as the hardware will allow.

• **MORE TASKS RUN SIMULTANEOUSLY**

because Idris requires less memory. Typically, 50 KB for the Kernel plus 50 KB for a compile. For example, you can overlap communication simultaneously with word processing and spreadsheet analysis and Fortran compilation.

• **EASY END-USER LICENSING**

provided by use of Whitesmiths' authorization seal.

That's UNIX with change. To get more out of your computer, call or write to Whitesmiths, Ltd.

Whitesmiths, Ltd.

97 Lowell Road Concord, MA 01742
(617) 369-8499 Telex 951708 SOFTWARE CNCM

Circle no. 67 on reader service card.

UNIX is a trademark of Bell Laboratories; DEC, PDP-11 and Rainbow are trademarks of Digital Equipment Corporation; IBM and IBM PC are trademarks of International Business Machines Corporation; MS-DOS is a trademark of Microsoft Corp.; Idris is a trademark of Whitesmiths, Ltd. Distributors: Australia, Fawnray Pty. Ltd., Hurstville, (612) 570-6100; Japan, Advanced Data Controls Corp., Chiyoda-ku, Tokyo (03) 263-0383; United Kingdom, Real Time Systems, Douglas, Isle of Man 0624833403; Sweden, Unisoft A.B., Goteborg, 031-13-56-32.

ENTER... THE FUTURE!

TURBO PASCAL

Introductory offer
\$49.95

**THIS IS THE PASCAL COMPILER
EVERYBODY'S BEEN WAITING FOR...
EVERYBODY EXCEPT THE COMPETITION!**

Extended Pascal for your IBM PC, APPLE CP/M, MS DOS, CP/M 86, CCP/M 86 or CP/M 80 computer features:

- Full screen interactive editor providing a complete menu driven program development environment.
- 11 significant digits in floating point arithmetic.
- Built-in transcendental functions.
- Dynamic strings with full set of string handling features.
- Program chaining with common variables.
- Random access data files.
- Full support of operating system facilities.
- And much more.

ORDER YOUR COPY OF **TURBO PASCAL** TODAY TO TAKE ADVANTAGE OF OUR INTRODUCTORY SPECIAL.

For Visa and MasterCard orders call toll free

1-800-227-2400 X 968

IN CA: 1-800-772-2666 X 968
(lines open 24 hrs. a day, 7 days a week)

Dealer & Distributor Inquiries welcome.

	Turbo Pascal	IBM Pascal	Pascal MT+
PRICE	49.95	300.00	595.00
Compile & Link speed	1 second!!!	97 seconds	90 seconds
Execution speed	2.2 seconds	9 seconds	3 seconds
Disk Space 16 bit 8 bit	33K w editor! 28K w editor!	300K + editor Not Available	225K + editor 168K + editor
8 and 16 bit	YES	NO	YES
built-in editor	YES	NO	NO
Generate object code	YES	YES	YES
One pass native code compiler	YES	NO	NO
Locates RunTime errors directly in source code	YES	NO	NO

Benchmark data based on EightQueens in "Algorithms + Data Structures - Programs" by N. Wirth, run on an IBM PC.

Turbo Pascal is a trademark of Borland International. MT+ is a trademark of MT MicroSystems. IBM is a trademark of International Business Machines.

LEARN TO WRITE A SPREADSHEET

Our Introductory offer includes **MICROCALC**, a spreadsheet written in **Turbo Pascal**. It will be on your disk, and ready to run. And we've included the source code to show you exactly how a spreadsheet is written!

Turbo Pascal includes a 250 page bound manual with extensive explanations and many illustrative examples.

Turbo Pascal \$49.95 + \$5.00 shipping per copy.

Check _____ Money Order _____
VISA _____ MasterCard _____

Card #: _____
Exp date: _____ Shipped UPS

BORLAND INTERNATIONAL

Borland International
4807 Scotts Valley Drive
Scotts Valley, California 95066

My system is: 8 bit _____ 16 bit _____

Operating system: CP/M 80 _____

CP/M 86 _____ MS DOS _____ PC DOS _____

Computer: _____ Disk Format: _____

Please be sure model number and format are correct.

NAME: _____

ADDRESS: _____

CITY/STATE/ZIP: _____

TELEPHONE: _____

California residents add 6 1/2% sales tax. Outside North America add \$15.00. Checks must be on a U.S. bank, and in U.S. dollars. Sorry, no C.O.D.